

DISEÑO DE UN CRIPTOPROCESADOR RSA DE 8192 BITS USANDO UN ARREGLO SISTÓLICO

CLAUDIA PATRICIA RENTERÍA MEJÍA

Tesis presentada a la Universidad del Valle

En cumplimiento parcial de los requisitos para el grado de

Maestría en Ingeniería

Con énfasis en Ingeniería Electrónica

UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

SANTIAGO DE CALI, FEBRERO DE 2012

DISEÑO DE UN CRIPTOPROCESADOR RSA DE 8192 BITS USANDO UN ARREGLO SISTÓLICO

CLAUDIA PATRICIA RENTERÍA MEJÍA

Tesis presentada a la Universidad del Valle

En cumplimiento parcial de los requisitos para el grado de

Maestría en Ingeniería

Con énfasis en Ingeniería Electrónica

Aprobada:

Edward Moreno, PhD

Universidad Federal de Sergipe, Brasil

Paulo César Realpe, MSc

Institución Universitaria

Antonio José Camacho

Jaime Velasco Medina, PhD

Director

Rubén Darío Nieto, PhD

Coordinador Maestría en
Ingeniería con énfasis en
Electrónica

Dedicado a:

Mis padres, Gladys Mejía y Luis Rentería, por su apoyo incondicional

Mi sobrino, Sebastián Gutiérrez, la adoración de mi vida

AGRADECIMIENTOS

Le doy gracias a Dios por darme la oportunidad de volver a estudiar en la Universidad del Valle.

Le doy gracias a mis padres, Gladys y Luis, mi hermana, Paola Andrea, y mi sobrino, Sebastián, por su amor, compañía y apoyo en cada momento de mi vida.

Le doy gracias a mi director Jaime Velasco Medina por su confianza, apoyo y enseñanzas a lo largo de la maestría.

Le doy gracias a mis compañeros de grupo John Michael Espinosa Durán y Vladimir Trujillo Olaya, por sus enseñanzas, consejos, asesorías, valiosos aportes a mi tesis y el apoyo incondicional que me brindaron cuando más lo necesitaba.

Le doy gracias a Edward Moreno, PhD, y Paulo César Realpe, MSc, por aceptar ser jurados de mi tesis de maestría.

TABLA DE CONTENIDOS

Resumen.....	xx
--------------	----

CAPÍTULO 1

INTRODUCCIÓN

1.1. Motivación.....	1
1.2. Contribuciones de esta tesis.....	2
1.3. Organización de esta tesis.....	3

CAPÍTULO 2

ALGORITMOS PARA LA MULTIPLICACIÓN Y LA EXPONENCIACIÓN MODULAR

2.1. Introducción	5
2.2. Principios básicos de la aritmética modular	6
2.2.1. Definición de la aritmética modular.....	6
2.2.2. Operaciones de la aritmética modular.....	7

2.2.2.1. Adición y substracción modular.....	7
2.2.2.2. Multiplicación modular.....	8
2.2.2.3. División modular.....	9
2.2.2.4. Exponenciación modular.....	9
2.3. Encriptación y desencriptación en RSA	10
2.3.1. Algoritmo de Montgomery base r	11
2.3.2. Algoritmos de la exponenciación binaria <i>MSB-first</i>	14
2.3.2. Algoritmos de la exponenciación binaria <i>LSB-first</i>	14
2.3.3. Algoritmo de la exponenciación m-aria.....	15
2.3.4. Algoritmo de la exponenciación modular con ventana deslizante....	15

CAPÍTULO 3

DISEÑO DE LOS CRIPTOPROCESADORES RSA DE 8192 BITS BASADOS EN LOS MULTIPLICADORES DE MONTGOMERY BASE 2 Y 4

3.1. Diseño del multiplicador de Montgomery base 2 usando un arreglo sistólico	17
3.1.1. Algoritmo de Montgomery base 2.....	18
3.1.2. Algoritmo de Montgomery base 2 a nivel de bit.....	18

3.1.3. Arreglo 2D para el multiplicador de Montgomery base 2	19
3.1.4. Arreglo sistólico para el multiplicador de Montgomery base 2.....	21
3.1.5. Arquitectura del multiplicador de Montgomery base 2.....	23
3.2. Diseño del multiplicador de Montgomery base 4 usando un arreglo sistólico.....	26
3.2.1. Algoritmo de Montgomery base 4.....	26
3.2.2. Algoritmo de Montgomery base 4 a nivel de bit.....	27
3.2.3. Arreglo 2D para el multiplicador de Montgomery base 4 a nivel de bit.....	28
3.2.4. Arreglo sistólico para el multiplicador de Montgomery base 4.....	34
3.2.5. Arquitectura del multiplicador de Montgomery base 4.....	36
3.3. Selección del algoritmo para la exponenciación modular.....	40
3.4. Diseño del Criptoprocesador RSA	41
3.4.1. <i>Data path</i> del criptoprocesador RSA.....	42
3.4.2. Unidad de control del criptoprocesador RSA.....	43

CAPÍTULO 4

RESULTADOS DE VERIFICACIÓN Y SÍNTESIS PARA LOS MULTIPLICADORES DE MONTGOMERY Y LOS CRIPTOPROCESADORES RSA

4.1. Simulación funcional de los algoritmos de Montgomery y exponenciación binaria.....	46
4.1.1. Simulación del algoritmo de Euclides extendido.....	47
4.1.2. Simulación del algoritmo de Montgomery.....	47
4.1.3. Simulación del algoritmo de Montgomery base 2.....	48
4.1.4. Simulación del algoritmo de Montgomery base 4.....	50
4.1.5. Simulación de la exponenciación binaria.....	52
4.2. Verificación en hardware de los Criptoprocesadores RSA.....	54
4.2.1. Verificación en hardware del Criptoprocesador RSA basado en el multiplicador de Montgomery base 2.....	54
4.2.2. Verificación en hardware del Criptoprocesador RSA basado en el multiplicador de Montgomery base 4.....	63
4.3. Resultados de síntesis.....	70

CAPÍTULO 5

CONCLUSIONES Y TRABAJO FUTURO

5.1. Conclusiones.....	82
5.2. Trabajo futuro.....	84

BIBLIOGRAFÍA.....	85
-------------------	----

ANEXOS

Anexo 1: Datos para realizar la prueba de la exponenciación modular de 8192 bits.....	87
---	----

Anexo 2: Verificación hardware usando SignalTap del criptoprocador RSA de 8192 bits.....	89
--	----

LISTA DE ALGORITMOS

Algoritmo 2.1: Algoritmo de Euclides extendido.....	9
Algoritmo 2.2: Multiplicación Montgomery base r	12
Algoritmo 2.3: Determinación de $-N^{-1}$ usando el algoritmo de Euclides extendido.....	13
Algoritmo 2.4: Exponenciación binaria MSB first.....	14
Algoritmo 2.5: Exponenciación binaria LSB first.....	15
Algoritmo 2.6: Exponenciación m -aria.....	16
Algoritmo 2.7: Exponenciación usando el método de la ventana deslizante ...	16
Algoritmo 3.1: Multiplicación de Montgomery base 2	18
Algoritmo 3.2: Multiplicación de Montgomery base 2 a nivel de bit.....	19
Algoritmo 3.3: Multiplicación de Montgomery base 4	27
Algoritmo 3.4: Multiplicación de Montgomery base 4 a nivel de bit.....	28

LISTA DE FIGURAS

Figura 2.1: Modelo jerárquico de seis capas para aplicaciones de seguridad informática.....	6
Figura 3.1: Elemento de procesamiento P_0 , para $j=0$	20
Figura 3.2: Elemento de procesamiento P_j , para $j \neq 0$	20
Figura 3.3: Arreglo 1D para una iteración de i	21
Figura 3.4: Arreglo paralelo 2D para el algoritmo de Montgomery base 2 a nivel de bit, $n = 4$ bits, A y B tiene 5 bits.....	22
Figura 3.5: Arreglo sistólico para el multiplicador de Montgomery base 2, $n = 4$ bits.....	23
Figura 3.6: Data path para el multiplicador de Montgomery base 2, $n = 4$ bits.....	24
Figura 3.7: Máquina de estados algorítmica para el arreglo sistólico del multiplicador de Montgomery base 2.....	25
Figura 3.8: Multiplicador de Montgomery base 2: control path y data path.....	25
Figura 3.9: Sumador carry-save de cinco entradas y tres salidas, CSA53.....	29
Figura 3.10: Sumador CSA53 de cuatro bits.....	29

Figura 3.11: Elemento de procesamiento para $j=0$, PE_0	30
Figura 3.12: Elemento de procesamiento para $j \neq 0$, PE_j	31
Figura 3.13: Bloque funcional QS para $N'=3$	32
Figura 3.14: Bloque funcional CS_1.....	33
Figura 3.15: Arreglo 1D para la iteración i , $n=4$ dígitos base 4 y $nb=8$ bits.....	33
Figura 3.16: Arreglo 2D para el algoritmo de Montgomery base 4 a nivel de bit, $n =$ 4 dígitos base 4 ó $nb = 8$ bits (para el módulo)....	34
Figura 3.17: Arreglo sistólico para el multiplicador de Montgomery base 4, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).....	35
Figura 3.18: Arreglo sistólico del multiplicador de Montgomery base 4 que realiza los productos AxB_0 y AxB_1	36
Figura 3.19: Multiplicador por 3, Multx3, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).....	36
Figura 3.20: Multiplicador modular de Montgomery base 4, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).....	37
Figura 3.21: Máquina de estados algorítmica del multiplicador de Montgomery base 4.....	39
Figura 3.22: Multiplicador de Montgomery base 4, $nb = 8$ bits.....	39
Figura 3.23: Diagrama de bloques del criptoprocador RSA.....	42

Figura 3.24: Data path del criptoprocador RSA usando el multiplicador de Montgomery base 4.....	43
Figura 3.25: Bloque SEL_A.....	43
Figura 3.26: ASM de la máquina de estados del criptoprocador RSA usando el multiplicador de Montgomery base 4.....	44
Figura 3.27: Data path y control path del criptoprocador RSA usando el multiplicador de Montgomery base 4, $nb = 8$ bits.....	45
Figura 4.1: Procedimiento algeuext para simular en Maple el algoritmo de Euclides extendido.....	47
Figura 4.2: Procedimiento MMM para simular en Maple el producto de Montgomery.....	48
Figura 4.3: Resultados del procedimiento MMM.....	48
Figura 4.4: Procedimiento hrmontmult para la simulación en Maple del algoritmo de la multiplicación de Montgomery base 2....	49
Figura 4.5: Resultados del procedimiento para la multiplicación de Montgomery base 2.....	50
Figura 4.6: Procedimiento hrmontmult para simular en Maple el algoritmo de la multiplicación de Montgomery base 4.....	51
Figura 4.7: Resultados del procedimiento para la multiplicación de Montgomery base 4.....	52

Figura 4.8: Procedimiento exponenciación para la simulación en Maple del algoritmo de la exponenciación binaria.....	53
Figura 4.9: Resultados del procedimiento exponenciación para la exponenciación binaria.....	54
Figura 4.10: Resultados de la conversión hexadecimal-decimal del módulo N usando Maple.....	55
Figura 4.11: Resultados de la encriptación de un mensaje de 1024 bits usando Maple... ..	55
Figura 4.12: Resultados de la verificación en hardware del criptoprocesador RSA basado en el multiplicador de Montgomery base 2 para la encriptación de un mensaje de 1024 bits.....	56
Figura 4.13: Resultado de la conversión hexadecimal-decimal del exponente d usando Maple.....	56
Figura 4.14: Resultado de la desencriptación de un mensaje de 1024 bits usando Maple.....	57
Figura 4.15: Resultados de la verificación en hardware del criptoprocesador RSA basado en el multiplicador de Montgomery base 2 para la desencriptación de un mensaje de 1024 bits.....	58
Figura 4.16: Resultados de la conversión hexadecimal-decimal del exponente d usando Maple.....	59

Figura 4.17: Resultados de la conversión hexadecimal-decimal del módulo N usando Maple.....	60
Figura 4.18: Resultados de la exponenciación modular de un número de 8192 bits usando Maple.....	61
Figura 4.19: Conversión decimal-hexadecimal del resultado de la exponenciación modular de 8192 bits.....	62
Figura 4.20: Resultado de la verificación en hardware del criptoprocador RSA basado en el multiplicador de Montgomery base 2 con la exponenciación modular de 8192 bits.....	63
Figura 4.21: Resultados de la conversión hexadecimal-decimal del módulo N usando Maple.....	64
Figura 4.22: Resultados de la conversión hexadecimal-decimal del exponente e usando Maple.....	64
Figura 4.23: Resultados de la encriptación de un mensaje de 2048 bits usando Maple.....	64
Figura 4.24: Resultado de la verificación en hardware del criptoprocador RSA basado en el multiplicador de Montgomery base 4 para la encriptación de un mensaje de 2048 bits.....	65
Figura 4.25: Resultado de la conversión hexadecimal-decimal del exponente d usando Maple.....	65

Figura 4.26: Resultado de la descriptación de un mensaje de 2048 bits usando Maple.....	66
Figura 4.27: Resultado de la verificación en hardware del criptoprocador RSA basado en el multiplicador de Montgomery base 4 para la descriptación de un mensaje de 2048 bits.....	67
Figura 4.28: Resultado de la exponenciación modular de 8192 bits usando Maple.....	68
Figura 4.29: Resultado de la conversión de la exponenciación modular de 8192 bits a hexadecimal usando Maple.....	69
Figura 4.30: Área vs tamaño de clave para los multiplicadores de Montgomery base 2 y base 4.....	72
Figura 4.31: Frecuencia máxima de operación vs tamaño de clave para los multiplicadores de Montgomery base 2 y base 4.....	72
Figura 4.32: Área vs tamaño de clave para los criptoprocadores RSA..	74
Figura 4.33: Frecuencia máxima de operación vs tamaño de la clave para los criptoprocadores RSA.....	74
Figura 4.34: Tiempo de ejecución vs tamaño de la clave para la multiplicación de Montgomery.....	76
Figura 4.35: Tiempo de ejecución vs tamaño de clave para la exponenciación modular.....	77

Figura 4.36: Tiempo de ejecución para la descriptación usando Maple, y los criptoprocesadores RSA basados en el MMB2 y el MMB4.....	78
Figura 4.37: Área vs. Tamaño de clave para el MMB2, el MMB4 y los dos criptoprocesadores RSA.....	79
Figura 4.38: Frecuencia máxima de operación vs. Tamaño de clave para el MMB2, el MMB4 y los dos criptoprocesadores RSA.....	80

LISTA DE TABLAS

Tabla 3.1: Promedio de multiplicaciones para diferentes algoritmos de exponenciación modular.....	41
Tabla 4.1: Resultados de síntesis del multiplicador de Montgomery base 2 para diferentes tamaños de clave en el FPGA EP3SL150F1152C2.....	77
Tabla 4.2: Resultados de síntesis del multiplicador de Montgomery base 4 para diferentes tamaños de clave en el FPGA EP3SL150F1152C2.....	77
Tabla 4.3: Resultados de síntesis del criptoprocesador RSA basado en el multiplicador de Montgomery base 2 para diferentes tamaños de clave en el FPGA EP3SL150F1152C2.....	79
Tabla 4.4: Resultados de síntesis del criptoprocesador RSA basado en el multiplicador de Montgomery base 4 para diferentes tamaños de clave en el FPGA EP3SL150F1152C2.....	79
Tabla 4.5: Tiempo de ejecución para la multiplicación de Montgomery base 2 y 4 usando la FMO para diferentes tamaños de clave.....	81
Tabla 4.6: Tiempo de ejecución para la exponenciación modular (desencriptación) de los criptoprocesadores RSA basados en el MMB2 y el MMB4 usando la FMO para diferentes tamaños de clave.....	82

Tabla 4.7: *Tiempo de ejecución para la descriptación usando Maple, y los
criptoprocesadores RSA basados en el MMB2 y el MMB4..... 84*

Tabla 4.8: *Resultados de síntesis y “throughput” para el criptoprocesador RSA de
8192 bits..... 84*

RESUMEN

DISEÑO DE UN CRIPTOPROCESADOR RSA DE 8192 BITS USANDO UN ARREGLO SISTÓLICO

La seguridad de la información es un campo de investigación y desarrollo tecnológico orientado principalmente hacia aplicaciones militares y comerciales. Por ejemplo, el intercambio de información confidencial a través de Internet, tales como las transacciones bancarias y los servicios de telecomunicaciones son de gran impacto a través de una amplia gama de aplicaciones comerciales e industriales. Entonces, con el propósito de proteger los datos confidenciales en equipos informáticos y de comunicaciones, se deben usar criptosistemas, los cuales se pueden implementar en software y/o hardware. En este contexto, RSA es uno de los criptosistemas de clave pública más utilizados y tiene un buen nivel de seguridad si el tamaño de la clave es mayor o igual a 1024 bits. Sin embargo, RSA de 1024 bits se puede romper con la actual tecnología informática. Por lo tanto, es necesario aumentar el nivel de seguridad de los criptosistemas RSA usando claves de 2048, 4096 ó 8192 bits.

Desde el punto de vista matemático, RSA se basa en la exponenciación modular y ésta se lleva a cabo usando la multiplicación modular de Montgomery, la cual determina el desempeño de RSA debido a que esta multiplicación utiliza la adición de tres datos muy grandes.

Teniendo en cuenta las consideraciones anteriores para realizar una implementación en hardware de un criptoprocador RSA de 8192 bits, es necesario usar el multiplicador de Montgomery basado en un arreglo sistólico usando sumadores “carry-save”. Entonces, en este trabajo se presenta el diseño de dos criptoprocadores RSA de 8192-bits, los cuales son diseñados usando el algoritmo de exponenciación binaria y los algoritmos de Montgomery para base-2

y base-4. En este caso, los multiplicadores diseñados realizan simultáneamente dos multiplicaciones, lo que permite aumentar el “throughput” de los criptoprocesadores.

CAPÍTULO 1

INTRODUCCIÓN

1.1. MOTIVACIÓN

Durante los últimos años, la seguridad de la información se ha convertido en un campo técnico-científico muy importante no sólo para las instituciones militares o gubernamentales, sino también para el sector de los negocios y las personas del común [1]. El intercambio de información confidencial a través de la internet, tales como transacciones bancarias, claves de tarjetas de crédito y servicios de telecomunicaciones se han convertido en prácticas comunes, es decir, a medida que el mundo se encuentre más comunicado la dependencia de los servicios electrónicos va aumentando. Entonces, con el propósito de proteger datos confidenciales en equipos de cómputo y sistemas de comunicación se deben adoptar medios confiables y no interceptables para almacenar y transmitir datos, lo cual se puede lograr usando algún tipo de criptosistema.

En este contexto, RSA es el criptosistema de clave pública más usado en aplicaciones que requieren de una buena seguridad en la información. Este criptosistema se puede implementar tanto a nivel de software (la gran mayoría de las aplicaciones) como a nivel de hardware. Sin embargo, las implementaciones hardware de Criptosistemas permiten obtener un mayor nivel de seguridad y una mayor velocidad de procesamiento de los datos que su contraparte en software.

Entonces, teniendo en cuenta el actual flujo de información y el estado del arte de la tecnología para el procesamiento computacional, es necesario desarrollar

implementaciones en hardware de criptosistemas que suministren el mayor nivel de seguridad. Sin embargo, en la literatura consultada, la mayoría son implementados para claves de 512, 1024 y 2048 bits [2],[3],[4].

Ante esta realidad y considerando las futuras demandas de alta seguridad es necesario diseñar criptoprocesadores en hardware con tamaños de clave mayores a 1024 bits para los criptosistemas RSA. Por lo tanto, esta tesis presenta el diseño de criptoprocesadores RSA para tamaños de clave de 2048, 4096 y 8192 bits, los cuales presentan alto *'throughput'*, usan mínima área, y pueden ser integrados en un criptosistema embebido.

1.2. CONTRIBUCIONES DE ESTA TESIS

Actualmente, el criptosistema RSA es el más utilizado en aplicaciones que requieren alta seguridad tal como el comercio electrónico, y su nivel de seguridad se basa en el problema de factorización de enteros de gran tamaño [5].

La operación aritmética usada en el criptosistema RSA para encriptar y desencriptar mensajes es la exponenciación modular, la cual usa varias multiplicaciones modulares. En este caso, para llevar a cabo la multiplicación modular, el algoritmo de Montgomery es una solución eficiente, el cual permite realizar la suma de datos de gran tamaño. Entonces, la implementación de la operación de la suma para datos de gran tamaño genera una muy larga cadena de acarreos, lo cual degrada significativamente el *'throughput'* del criptosistema para claves mayores a 1024 bits. Con el propósito de mitigar la dificultad anterior la operación de la suma es llevada a cabo usando *"carry save adders"*, para reducir el tiempo de propagación.

De otro lado, la implementación hardware de sumadores considerando datos de gran tamaño es una tarea difícil debido a que es muy complejo lograr un buen balance entre los parámetros de desempeño, tales como velocidad, potencia y área. Por lo tanto, el algoritmo de Montgomery se mapea en un arreglo sistólico con el propósito de minimizar el área del multiplicador y eliminar las señales de tipo global. Adicionalmente, se debe considerar un algoritmo eficiente para la

implementación en hardware de la exponenciación modular, usando el multiplicador de Montgomery.

Otro desafío en los Criptosistemas RSA es usar claves de mayor tamaño debido a que actualmente, con técnicas de criptoanálisis, se han reportado ataques exitosos a criptosistemas con claves de hasta 1024 bits [6].

Teniendo en cuenta las consideraciones anteriores y los desafíos que actualmente enfrentan los criptosistemas RSA, en esta tesis se diseñan dos criptoprocesadores para el criptosistema RSA que brindan un nivel muy alto de seguridad, es decir, usan una clave mayor a 1024 bits. En este contexto las principales contribuciones alcanzadas en el desarrollo de esta tesis son:

- Diseño de un multiplicador de Montgomery base 2 usando un arreglo sistólico para claves de 1024, 2048, 4096, 8192 bits.
- Diseño de un multiplicador de Montgomery base 4 usando un arreglo sistólico para claves de 1024, 2048, 4096, 8192 bits.
- Diseño de un criptoprocesador RSA usando el algoritmo de exponenciación binaria y el multiplicador de Montgomery base 2.
- Diseño de un criptoprocesador RSA usando el algoritmo de exponenciación binaria y el multiplicador de Montgomery base 4.

Adicionalmente, los dos multiplicadores pueden realizar dos productos de forma paralela, lo cual es de mucha utilidad para llevar a cabo de forma eficiente el algoritmo de exponenciación binaria. Con los otros algoritmos de exponenciación modular esta ventaja no se puede aprovechar, siendo este uno de los factores que determinó la selección del algoritmo de exponenciación binaria para el diseño del criptoprocesador RSA.

1.3. ORGANIZACIÓN DE ESTA TESIS

Esta tesis está dividida en cinco capítulos, los cuales están organizados de la siguiente forma:

En el capítulo 2 se presentan los niveles jerárquicos para aplicaciones de seguridad informática, los principios básicos de la aritmética modular, los procedimientos para realizar la operación de encriptación en el criptosistema de clave pública RSA, y los algoritmos para la multiplicación y la exponenciación modular.

En el capítulo 3 se describen los diseños de los multiplicadores de Montgomery base 2 y base 4. Finalmente se presenta el diseño de los dos criptoprocesadores RSA con base en el algoritmo de exponenciación binaria usando los multiplicadores de Montgomery base 2 y base 4, respectivamente.

En el capítulo 4 se presentan los procedimientos implementados en Maple para los algoritmos de Montgomery y exponenciación binaria. También se presentan y analizan los resultados de síntesis de los multiplicadores de Montgomery base 2 y base 4, y los resultados de síntesis de los dos criptoprocesadores RSA para tamaños de clave de 512, 1024, 2048, 4096, 8192 bits. Finalmente se presentan los resultados de la verificación del hardware usando SignalTap II y la comparación de estos resultados con los obtenidos usando Maple.

En el capítulo 5 se presentan las conclusiones y las actividades de investigación para el trabajo futuro.

CAPÍTULO 2

ALGORITMOS PARA LA MULTIPLICACIÓN Y LA EXPONENCIACIÓN MODULAR

2.1. INTRODUCCIÓN

El uso de las técnicas criptográficas permite alcanzar, en los sistemas de computación, un nivel de seguridad en la transferencia y almacenamiento de la información. Las aplicaciones que ofrecen este tipo de servicio, conocidas como de seguridad en la información, pueden ser estudiadas con base en el modelo jerárquico de seis capas presentado en la Figura 2.1 [1]. Cada capa es soportada por la capa del nivel inmediatamente inferior, y los niveles usados para llevar a cabo el desarrollo de esta tesis, son los correspondientes a los rectángulos de color gris (niveles 1,2 y 3).

Como se puede observar en la Figura 2.1, específicamente en el nivel 2, hay dos tipos de criptografía. Cada una de ellas se encuentra soportada por un algoritmo basado en operaciones aritméticas. En este caso, este trabajo es orientado hacia la criptografía de clave asimétrica, es decir el diseño de un procesador para el criptosistema RSA. Con respecto al nivel 3, las primitivas criptográficas a implementar son la encriptación y desencriptación. En lo relacionado con el nivel 1, la criptografía RSA usa la aritmética modular, llevando a cabo las primitivas criptográficas de encriptación y desencriptación con la exponenciación modular.

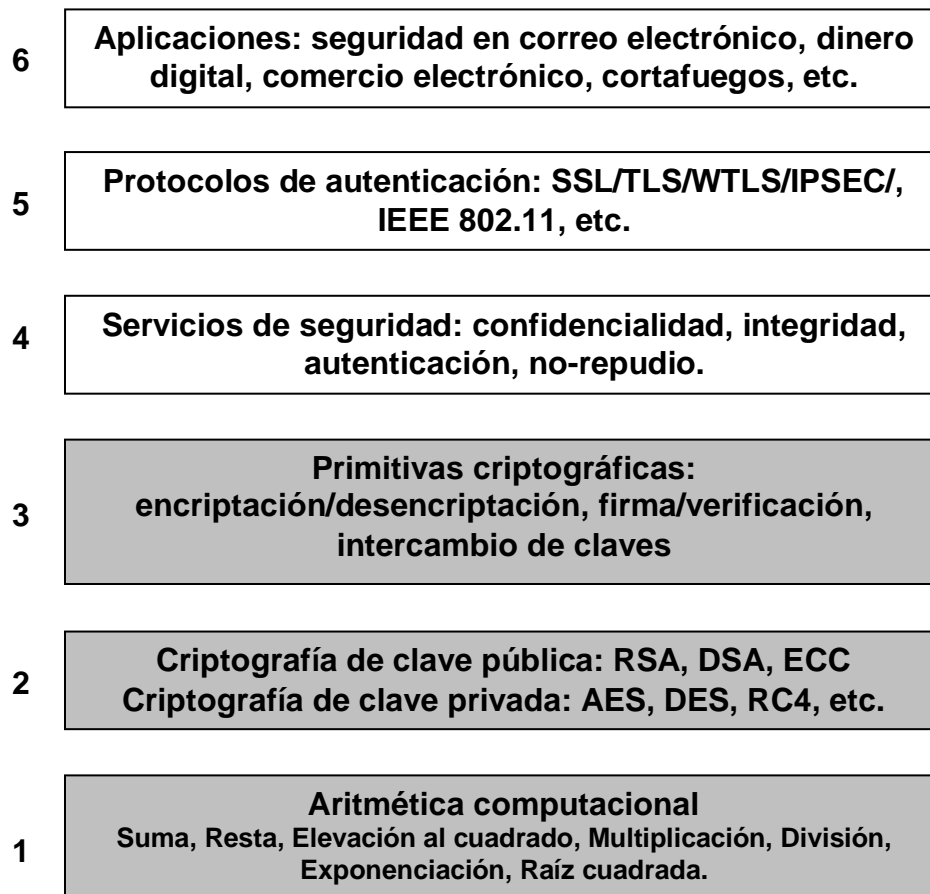


Figura 2.1. Modelo jerárquico de seis capas para aplicaciones de seguridad informática [1].

2.2. PRINCIPIOS BÁSICOS DE LA ARITMÉTICA MODULAR

En esta sección se describen los principios básicos de la aritmética modular de acuerdo con [1].

2.2.1. Definición de la aritmética modular

La aritmética modular se basa en la relación de congruencia, donde congruencia se define de la siguiente forma:

Dado $m \in \mathbb{Z}$, $m > 1$, se dice que $a, b \in \mathbb{Z}$ son congruentes módulo m si y solo si $m|(a-b)$, es decir m es un múltiplo de $a-b$ ó m es divisible por $a-b$. Esta relación se escribe como $a \equiv b \pmod{m}$. Donde m es el módulo de la congruencia. Si $m|(a-b)$ entonces a y b tienen el mismo residuo cuando son divididos entre m .

Se define \mathbb{Z}_m como el conjunto de todos los residuos positivos módulo m , es decir $\mathbb{Z}_m = \{0, 1, 2, \dots, m-1\}$.

Si $m \in \mathbb{N}$ y $a, b, c, d \in \mathbb{Z}$ tal que $a \equiv b \pmod{m}$ y $c \equiv d \pmod{m}$, entonces se cumplen las siguientes propiedades:

- $a + c \equiv b + d \pmod{m}$
- $a - c \equiv b - d \pmod{m}$
- $a \cdot c \equiv b \cdot d \pmod{m}$

La relación de la congruencia módulo m es una relación de equivalencia para todo $m \in \mathbb{Z}$. Sean $a, b, c \in \mathbb{Z}$, entonces la relación de congruencia satisface las siguientes propiedades:

1. Reflexiva: $a \equiv a \pmod{m}$
2. Simétrica: Si $a \equiv b \pmod{m}$ entonces $b \equiv a \pmod{m}$
3. Transitiva: Si $a \equiv b \pmod{m}$ y $b \equiv c \pmod{m}$ entonces $a \equiv c \pmod{m}$

2.2.2. Operaciones de la aritmética modular

2.2.2.1. Adición y substracción modular

Si $a, b \in \mathbb{Z}_m$ entonces se define la adición modular $a + b \pmod{m}$ como un elemento que pertenece a \mathbb{Z}_m . Las propiedades más importantes de la adición modular son:

1. Conmutativa, $a + b \pmod{m} = b + a \pmod{m}$
2. Asociativa, $(a+b)+c \pmod{m} = a+(b+c) \pmod{m}$
3. Tiene un elemento neutro (0), tal que $a + 0 = a \pmod{m}$

4. Para cada a y b en Z_m existe un único elemento x en Z_m tal que $a + x = b \bmod m$

Usando la última propiedad y $b = 0$, se puede decir que para cada a en Z_m , existe un único elemento x en Z_m , tal que $a + x \equiv 0 \bmod m$

2.2.2.2. Multiplicación modular

Si $a, b \in Z_m$, se define la multiplicación modular como, $c = a * b \bmod m$, donde c es un elemento en Z_m . Las propiedades más importantes de la multiplicación modular son:

1. Conmutativa, $a * b \bmod m = b * a \bmod m$
2. Asociativa, $(a * b) * c \bmod m = a * (b * c) \bmod m$
3. Tiene un elemento neutro (1), tal que $a * 1 = a \bmod m$
4. Si $\text{mcd}(m, c) = 1$ y $a * c = b * c \bmod m$, entonces $a \equiv b \bmod m$. Si m es un número primo, esta propiedad siempre se cumple.

Usando la última propiedad, se define el inverso multiplicativo de un número a de la siguiente forma:

Se dice que a tiene un inverso módulo m si existe un entero b tal que $1 \equiv a * b \bmod m$. Entonces, el entero b es el inverso de a y se representa como a^{-1} . El inverso de un número $a \bmod m$ existe si y sólo si existen dos números enteros x, y tal que $ax + my = 1$ y estos números existen si y sólo si $\text{mcd}(a, m) = 1$.

Con el objetivo de obtener el inverso modular de un número a se debe usar el algoritmo de Euclides extendido, presentado en el Algoritmo 2.1, con el cual es posible encontrar dos números enteros x, y que satisfagan la ecuación:

$$ax + my = 1 \tag{2.1}$$

Algoritmo 2.1: Algoritmo de Euclides Extendido

Entradas: a, b . a y b son enteros positivos. $a \geq b$

Salidas: d, x, y . $d = \text{mcd}(a, b)$. $a \cdot x + b \cdot y = d$

1. **if** $b = 0$ **then**
 2. $d := a; x := 1; y := 0;$
 3. **Return** (d, x, y)
 4. **end if**
 5. $x_1 := 0; x_2 := 1; y_1 := 1; y_2 := 0;$
 6. **while** $b > 0$ **do**
 7. $q := a \text{ div } b; r := a \bmod b;$
 8. $x := x_2 - qx_1; y := y_2 - qy_1;$
 9. $a := b; b := r; x_2 := x_1;$
 10. $x_1 := x; y_2 := y_1; y_1 := y;$
 11. **end while**
 12. $d := a; x := x_2; y := y_2;$
 13. **Return** (d, x, y)
-

2.2.2.3. División Modular

Usando las definiciones anteriores se dice que $a, b \in \mathbb{Z}_p$ y p es un número primo, se puede realizar la división a entre b calculando $a \cdot b^{-1} \bmod m$, donde b^{-1} es el inverso multiplicativo de b modulo p .

2.2.2.4. Exponenciación Modular

Se define la exponenciación modular, como el problema de calcular el número $b = a^e \bmod m$, con $a, b \in \mathbb{Z}_m$, y $e \in \mathbb{N}$. Observando que

$$x \cdot y \bmod m = [(x \bmod m) \cdot y \bmod m] \bmod m \quad (2.2)$$

Se concluye entonces que el problema de la exponenciación modular puede ser resuelto multiplicando números que nunca exceden el módulo m .

2.3. ENCRIPCIÓN Y DESENCRIPTACIÓN EN RSA

Las aplicaciones del criptosistema RSA son la encriptación y la firma digital [7]. En esta tesis se trabaja con la encriptación, por lo tanto, a continuación se da una breve descripción del procedimiento para realizar la encriptación y la desenscriptación en el criptosistema RSA, las cuales están basadas en el *PKCS # 1, V 2.1* [8].

De acuerdo con [8], en el criptosistema RSA se usan dos tipos de clave: clave pública y clave privada. La clave pública consiste de dos componentes: n y e , donde n es el módulo RSA y e es el exponente público RSA. Ambos datos son enteros positivos. La clave privada puede tener dos representaciones. En esta tesis se trabajará con la representación que consiste en el par n, d , donde n es el módulo RSA y d es el exponente privado RSA, ambos datos son enteros positivos.

Entre los componentes de la clave privada y la clave pública se deben cumplir unas relaciones matemáticas, las cuales son presentadas en esta sección con base en [1]. Estas relaciones se establecen con base en la obtención del módulo n , el cual debe ser el producto de dos primos grandes (su tamaño debe ser mayor o igual a 512 bits), es decir, $n = p * q$. El exponente público e es un número en el rango $1 < e < \phi(n)$ tal que, [1]

$$\text{mcd}(e, \phi(n)) = 1 \quad (2.3)$$

donde $\phi(n)$ es la función de Euler de n , dada por:

$$\phi(n) = (p-1)(q-1) \quad (2.4)$$

El exponente privado d es obtenido invirtiendo e módulo $\phi(n)$:

$$d = e^{-1} \text{ mod } \phi(n) \quad (2.5)$$

usando el algoritmo de Euclides extendido. Generalmente, se selecciona un exponente público pequeño, por ejemplo $e=2^{16} + 1$ [1]. Entonces la operación de encriptación es llevada a cabo calculando:

$$C = M^e \bmod n \quad (2.6)$$

donde M es el texto plano tal que $0 \leq M < n$, y C es el texto encriptado a partir del cual se puede obtener el texto plano M usando:

$$M = C^d \bmod n \quad (2.7)$$

Con base en lo anterior, se concluye que la operación fundamental en RSA es la exponenciación modular, la cual se efectúa realizando multiplicaciones modulares. Por lo tanto, a continuación se presentan los algoritmos para llevar a cabo de forma eficiente, en hardware, estas dos operaciones.

2.3.1. Algoritmo de Montgomery base r

El algoritmo usado para realizar la multiplicación modular es el de Montgomery, el cual se presenta en el Algoritmo 2.2 con base en la representación usada en [9] y el algoritmo presentado en [5].

La notación que se va a usar es la siguiente:

- Sea r la base, que es generalmente una potencia de 2.
- Los enteros se consideran con la representación base r , así:

$$A = \sum_{i=0}^n a_i r^i \quad (2.8)$$

donde $0 \leq a_i \leq r-1$.

- N es el módulo, el cual debe ser un número entero impar [1]. $N=(N_0, \dots, N_{n-1})_r$.
- A se puede ver como un vector con componentes en representación base r : $A = (a_0, \dots, a_n)_r$.

Algoritmo 2.2: Multiplicación modular de Montgomery base r

Entradas: $A = (a_0, \dots, a_n)_r \leq 2N - 1$ con $a_n \leq 1$, $B \leq 2N - 1$

Precálculo: $N' = -N^{-1} \bmod r$

Variable temporal: $T = (t_0, \dots, t_n)_r \leq 2N - 1$

Salida: $T = A * B * R^{-1} \bmod N$, $R = r^{n+1}$ y $T \leq 2N - 1$

1. $T := 0$;
 2. **for** $i=0$ **to** n **do**
 3. $Q := [t_0 + a_i b_0] * N' \bmod r$;
 4. $T := [T + a_i B + QN] \text{ div } r$;
 5. **end for**
 6. **Return** (T)
-

donde N' es el dígito menos significativo, en base r , de $-N^{-1} \bmod R$, el cual se obtiene determinando el inverso multiplicativo de $N \bmod R$, es decir $N^{-1} \bmod R$, usando el algoritmo de Euclides extendido y luego se calcula $-N^{-1}$ realizando operaciones de aritmética modular como se presenta en el Algoritmo 2.3.

El resultado obtenido al ejecutar el Algoritmo 2.2, T , es el producto de Montgomery, el cual puede ser calculado con la ecuación (2.9).

$$T = A * B * R^{-1} \bmod N \quad (2.9)$$

R^{-1} es el inverso multiplicativo de $R \bmod N$ [1].

El producto modular de A y B se obtiene con base en T usando (2.10), que es el producto Montgomery de T y R^2 .

$$P_{\bmod AB} = T * R^2 * R^{-1} \bmod N \quad (2.10)$$

Algoritmo 2.3: Determinación de $-N^{-1}$ usando el
Algoritmo de Euclides Extendido

Entradas: $R=r^{n+1}$, N :Módulo, n es el número bits del
módulo

Salida: $Neg_Inv_N = -N^{-1} \bmod R$

1. $a := R;$
 2. $b := N;$
 3. $x := 0;$
 4. $y := 1;$
 5. $lastx := 1;$
 6. $lasty := 0;$
 7. **While** $b \neq 0$ **do**
 8. $quotient := \text{floor}(a/b);$
 9. $temp := b;$
 10. $b := a \bmod b;$
 11. $a := temp;$
 12. $temp := x;$
 13. $x := lastx - quotient * x;$
 14. $lastx := temp;$
 15. $temp := y;$
 16. $y := lasty - quotient * y;$
 17. $lasty := temp;$
 18. **end while**
 19. $Neg_Inv_N \leftarrow -(lasty - R)$
 20. **Return** (Neg_Inv_N)
-

2.3.2. Algoritmo de la exponenciación binaria *MSB first*

El algoritmo para la exponenciación binaria *MSB first* se presenta en el Algoritmo 2.4 con base en [1] y [10], el cual se lleva a cabo calculando productos Montgomery. En este caso, *MMM* es la operación Multiplicación Modular de Montgomery.

Algoritmo 2.4: Exponenciación binaria *MSB first*

Entradas: Entero positivo M , $E = e_{k-1}e_{k-2}\dots e_0$, donde $e_i \in \{0,1\}$, módulo N de n bits,
 $R2 = R^*R \bmod N$, donde $R = r^{n+2}$

Salida: $C = M^E \bmod N$

1. $M' := MMM(M, R2)$; $C' := MMM(1, R2)$;
 2. **for** $i = k-1$ **down to** 0 **do**
 3. $C' := MMM(C', C')$;
 4. **if** $e_i = 1$ **then**
 5. $C' := MMM(M', C')$;
 6. **end if**
 7. **end for**
 8. $C := MMM(C', 1)$;
 9. **Return**(C);
-

2.3.2. Algoritmo de la exponenciación binaria *LSB first*

El algoritmo para la exponenciación binaria *LSB first* se presenta en el Algoritmo 2.5 con base en [10], el cual se lleva a cabo calculando productos Montgomery. En este caso, *MMM* es la operación Multiplicación Modular de Montgomery.

Algoritmo 2.5: Exponenciación Binaria *LSB first*

Entradas: Entero positivo M , $E = e_{k-1}e_{k-2}\dots e_0$, donde $e_i \in \{0,1\}$,
módulo N de n bits, $R2 = R^*R \bmod N$, donde $R = r^{n+2}$

Salida: $C = M^E \pmod{N}$

1. $M' := MMM(M, R2)$; $C' := MMM(1, R2)$;
 2. **for** $i=0$ **to** $k-1$ **do**
 3. **if** $e_i = 1$ **then**
 4. $C' := MMM(M', C')$;
 5. **end if**
 6. $M' := MMM(M', M')$;
 7. **end for**
 8. $C := MMM(C', 1)$;
 9. **Return** (C) ;
-

2.3.3. Algoritmo de la exponenciación m-aria

El algoritmo para la exponenciación m-aria es una mejora del algoritmo para la exponenciación binaria, al procesar r bits del exponente por iteración. Este algoritmo se presenta a continuación con base en [11].

2.3.4. Algoritmo de la exponenciación modular con ventana deslizante

El algoritmo para la exponenciación modular con el método de la Ventana Deslizante tiene como objetivo ejecutar menos multiplicaciones que el algoritmo para la exponenciación m-aria, particionando el exponente en palabras de diferente tamaño, de tal forma que se obtengan la mayor cantidad de palabras con todos los bits en cero. Este algoritmo se presenta con base en [11].

Algoritmo 2.6: Exponenciación m-aria

Entradas: Entero positivo M , exponente $e=(e_{k-1}, e_{k-2}, \dots, e_0)_2$ y un módulo N

Salida: $C = M^e \pmod{N}$

1. Precálculo
 2. Calcular y almacenar $M^w \pmod{N}$ para $w = 2, 3, \dots, m-1$.
 3. Descomponer e en palabras de r bits f_i para i desde 0 hasta $s-1$.
 4. $C := M^{f_{s-1}} \pmod{N}$
 5. **for** i **from** $s-2$ **downto** 0, **do**;
 6. $C := C^{2^r} \pmod{N}$;
 7. **If** $f_i \neq 0$ **then**
 8. $C := C * M^{f_i} \pmod{N}$;
 9. **end if**
 10. **end for**
 11. **Return** (C);
-

Algoritmo 2.7: Exponenciación modular usando el método de la Ventana Deslizante

Entradas: Entero positivo M , exponente $e=(e_{k-1}, e_{k-2}, \dots, e_0)_2$ y un módulo N

Salida: $C = M^e \pmod{N}$

1. Precálculo
 2. $C := M^{f_{p-1}} \pmod{N}$
 3. **for** i **from** $p-2$ **down to** 0,
 4. $C := C^{L^{(f_i)}} \pmod{N}$
 5. **If** $f_i \neq 0$ **then**
 6. $C := C * M^{f_i} \pmod{N}$
 7. **end if**
 8. **end for**
 9. **Return** (C);
-

CAPÍTULO 3

DISEÑO DE LOS CRIPTOPROCESADORES RSA DE 8192 BITS BASADOS EN LOS MULTIPLICADORES DE MONTGOMERY BASE 2 Y 4

La selección de la base para el diseño de un multiplicador de Montgomery determina el desempeño del multiplicador y la cantidad de recursos usados, debido a que un incremento en el tamaño de la base implica menos ciclos para llevar a cabo la multiplicación pero a su vez el diseño demandará más *hardware* [12]. Teniendo en cuenta lo anterior y debido a que el tamaño de la clave para el criptoprocador es de 8192 bits, en esta tesis se presenta el diseño de dos multiplicadores de Montgomery para las bases 2 y 4 usando un arreglo sistólico, los cuales presentan un buen compromiso *área-throughput*. Adicionalmente, se presenta el diseño de dos criptoprocadores RSA, usando el algoritmo de la exponenciación binaria y los multiplicadores de Montgomery base 2 y base 4.

3.1. DISEÑO DEL MULTIPLICADOR DE MONTGOMERY BASE 2 USANDO UN ARREGLO SISTÓLICO

En esta sección se presenta el diseño del multiplicador de Montgomery base 2 usando el procedimiento propuesto en [13], el cual consiste en el mapeo del Algoritmo de Montgomery base 2 en un arreglo sistólico. En este caso, el procedimiento se lleva a cabo mediante los siguientes pasos:

- Descripción del algoritmo de Montgomery base 2.

- Descripción del algoritmo de Montgomery base 2 a nivel de bit.
- Obtención del arreglo 2D correspondiente al algoritmo de Montgomery base 2 a nivel de bit.
- Mapear el arreglo 2D del algoritmo de Montgomery base 2 a nivel de bit en un arreglo sistólico 1D.

3.1.1. Algoritmo de Montgomery base 2

En el Algoritmo 3.1 se presenta el pseudocódigo para la multiplicación de Montgomery base 2. A y B son dos números enteros de $n+1$ bits, y N es el módulo de n bits. En este caso, se llevan a cabo $n+2$ iteraciones para garantizar que el valor de la multiplicación de Montgomery, T_{n+1} , esté entre 0 y $2N-1$. Por lo tanto $a_{n+1} = 0$.

Algoritmo 3.1: Multiplicación de Montgomery base 2

Entradas: $A = (a_0, \dots, a_n)_2 \leq 2N - 1$, $B = (B_0, \dots, B_n)_2 \leq 2N - 1$, $N = (N_0, \dots, N_{n-1})_2$

Variable temporal: $T_i = (t_0, \dots, t_n)_2 \leq 2N - 1$

Salida: $T_{n+1} = A * B * R^{-1} \bmod N$, $R = r^{n+2}$ y $T_{n+1} \leq 2N - 1$

1. $T_{-1} := 0$
 2. **for** $i=0$ **to** $n+1$
 3. $Q_i := [T_{i-1} + a_i B] \bmod 2$;
 4. $T_i := [T_{i-1} + a_i B + Q_i N] \div 2$;
 5. **end for**
 6. **Return** (T_{n+1})
-

3.1.2. Algoritmo de Montgomery base 2 a nivel de bit

Para llevar a cabo una implementación hardware del algoritmo de Montgomery base 2 usando procesamiento paralelo es necesario describir este algoritmo a nivel de bit y usar sumadores *carry-save*, tal como se presenta en el Algoritmo 3.2.

Algoritmo 3.2: Multiplicación de Montgomery base 2 a nivel de bit
(sumadores *carry-save*)

Entradas: $A = (a_0, \dots, a_n)_2 \leq 2N - 1$, $B = (B_0, \dots, B_n)_2 \leq 2N - 1$, $N = (N_0, \dots, N_{n-1})_2$

Salida: $T = A * B * R^{-1} \bmod N$

```

1.  for i=0 to n+1 do
2.    for j= 0 to n do
3.      if j=0 then
4.         $Q_i := (T_{ij} + C0_{ij} + C1_{ij} + a_i B_j) \bmod 2$ ;
5.      end if
6.       $T' := (T_{ij} + a_i B_j + C0_{ij}) \bmod 2$ ;
7.       $C0_{i+1,j} := (T_{ij} + a_i B_j + C0_{ij}) \text{div} 2$ ;
8.       $T_{i+1,j-1} := (T' + Q_i N_j + C1_{ij}) \bmod 2$ ;
9.       $C1_{i+1,j} := (T' + Q_i N_j + C1_{ij}) \text{div} 2$ ;
10.    end for
11.  end for
12.  $T := T_{n+1} + C0_{n+1} + C1_{n+1}$ 

```

3.1.3. Arreglo 2D para el multiplicador de Montgomery base 2

La implementación paralela del Algoritmo 3.2 permite obtener un arreglo 2D, debido a que cada iteración del bucle j genera un elemento de procesamiento del producto parcial de Montgomery i , P_j , mientras que cada iteración del bucle i genera un producto parcial de Montgomery, T_i , $C0_i$ y $C1_i$.

En este caso, cada iteración del bucle j realiza la suma de T_{ij} , $C0_{ij}$, $C1_{ij}$, $a_i B_j$ y $Q_i N_j$ usando un sumador CSA de cinco entradas y tres salidas, generando los acarreos $C0_{i+1,j}$ y $C1_{i+1,j}$, y el bit del resultado $T_{i+1,j-1}$. En la iteración $j = 0$ se calcula Q_i con base en el paso 4 del Algoritmo 3.2. Entonces, se puede deducir que hay dos tipos de elementos de procesamiento: $P0$, para $j = 0$ y P_j , para $j \neq 0$, los cuales se muestran en las Figuras 3.1 y 3.2, respectivamente, donde *FA* es un *Full-Adder*.

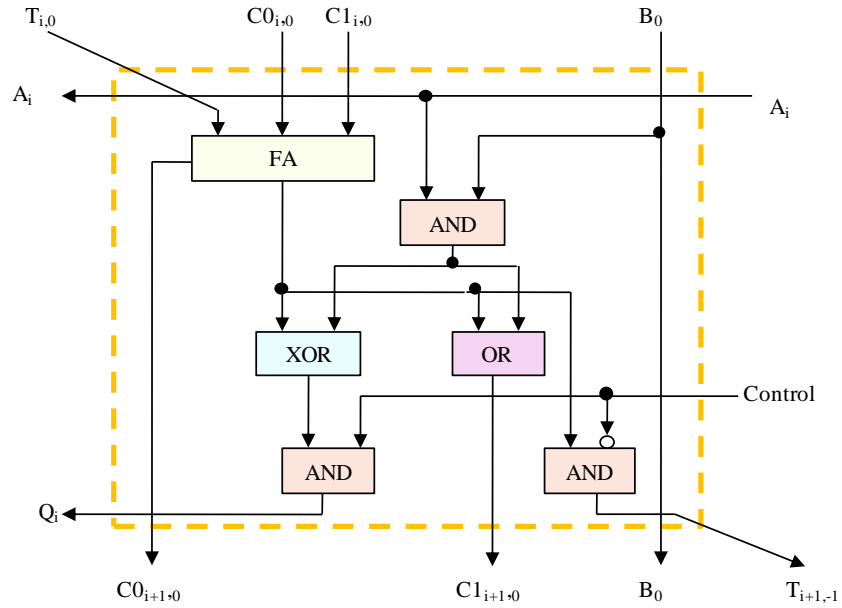


Figura 3.1. Elemento de procesamiento P_0 , para $j = 0$.

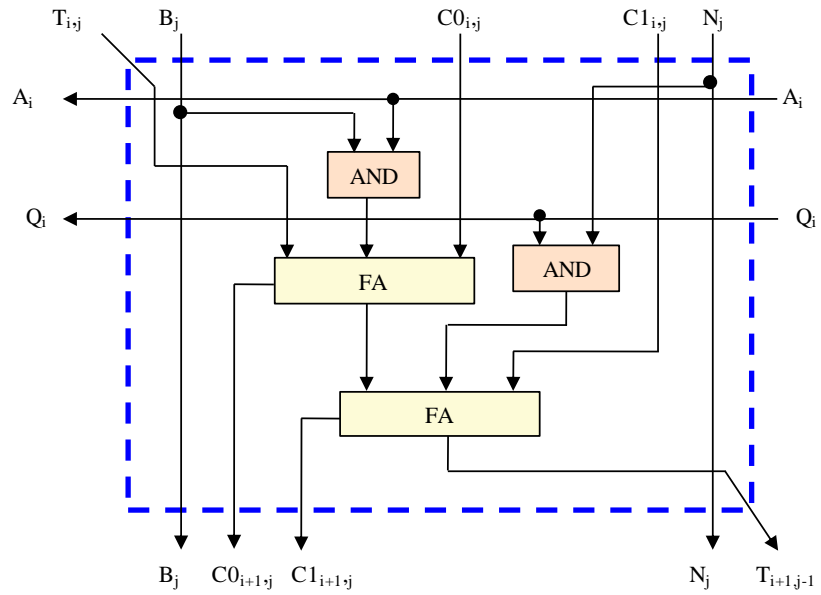


Figura 3.2. Elemento de procesamiento P_j , para $j \neq 0$.

En la Figura 3.3 se muestra el arreglo 1D generado por una iteración del bucle i para $n=4$ bits, es decir, j varía desde 0 hasta 4. Cada círculo representa una

iteración de j . Este arreglo 1D calcula el producto parcial de Montgomery para $a_i B$, es decir, la suma, T_{i+1} , y dos acarreos, $C0_{i+1}$ y $C1_{i+1}$.

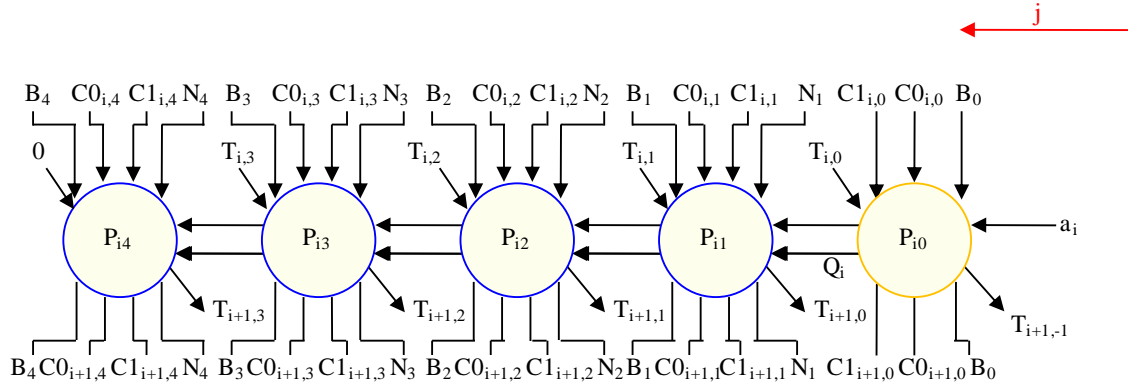


Figura 3.3. Arreglo 1D para una iteración de i .

En la Figura 3.4 se muestra el arreglo paralelo 2D obtenido a partir del Algoritmo 3.2, para $n = 4$ bits. Por lo tanto A y B tienen 5 bits, j varía desde 0 hasta 4 e i varía desde 0 hasta 5 para que el producto de Montgomery sea menor que $2N$. El dígito a_5 es cero. Cada fila representa una iteración de i , es decir, el producto parcial de Montgomery para $a_i B$. Cada círculo o elemento de procesamiento tiene un número en la parte inferior derecha, el cual indica el instante o ciclo de reloj en que procesa el elemento. Esta secuencia de procesamiento es fundamental para realizar el mapeo del arreglo paralelo 2D en un arreglo sistólico 1D.

3.1.4. Arreglo sistólico para el multiplicador de Montgomery base 2

En la Figura 3.5 se presenta el arreglo sistólico para el multiplicador de Montgomery base 2, el cual es obtenido a partir de la secuencia de procesamiento del arreglo paralelo 2D presentado en la Figura 3.4. El arreglo sistólico lleva a cabo el producto Montgomery en dos procesos: durante los primeros $2(n+2)$ ciclos la señal *Control* es '1', y se obtiene el valor de Q_i para cada bit de A ; y durante los últimos $2(n+2)$ ciclos la señal *Control* es '0' y se obtiene un bit de la multiplicación de Montgomery T_{i+1} cada dos ciclos de reloj, desde el bit menos significativo hasta el bit más significativo.

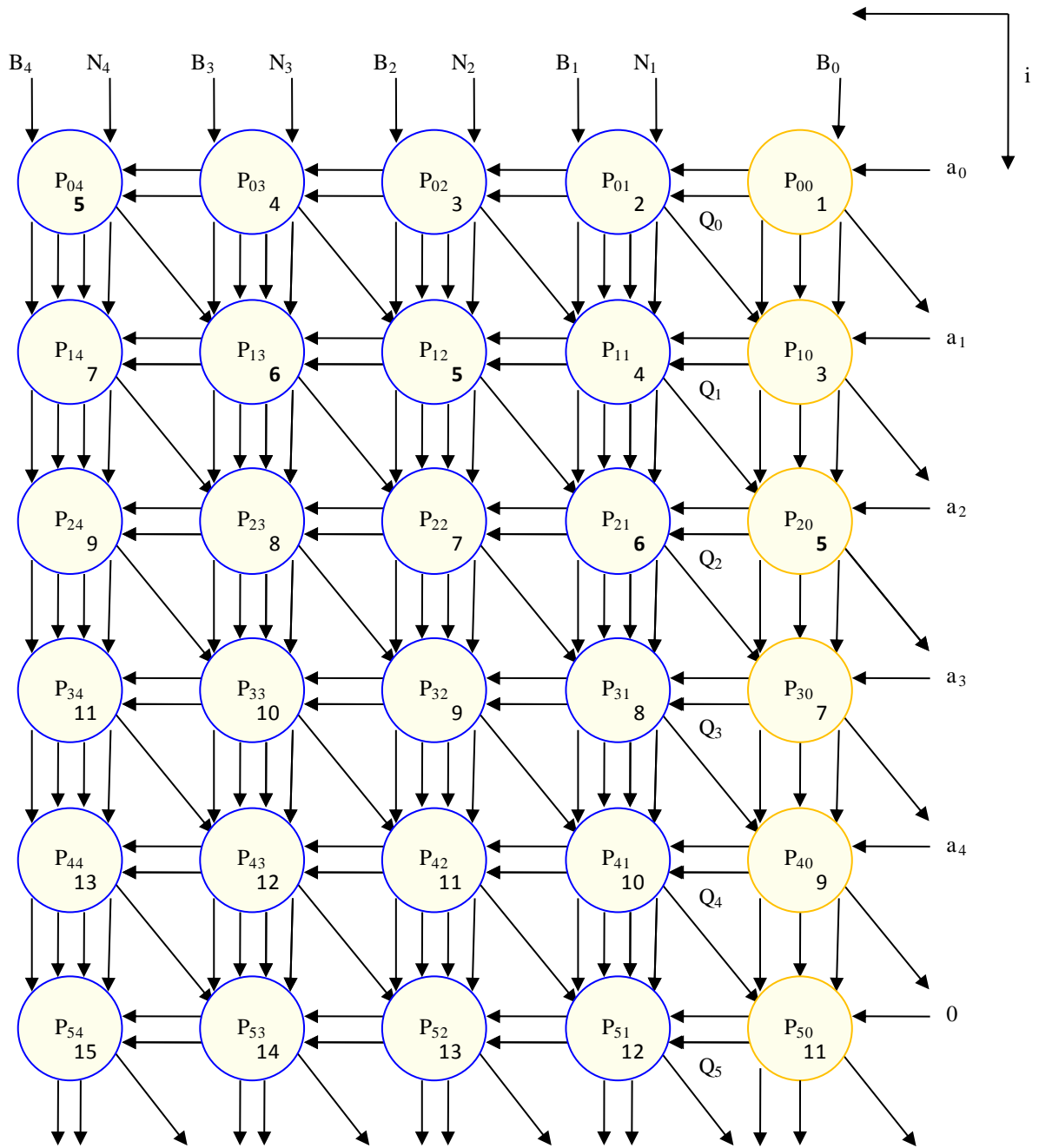


Figura 3.4. Arreglo paralelo 2D para el algoritmo de Montgomery base 2 a nivel de bit, $n = 4$ bits, A y B tienen 5 bits.

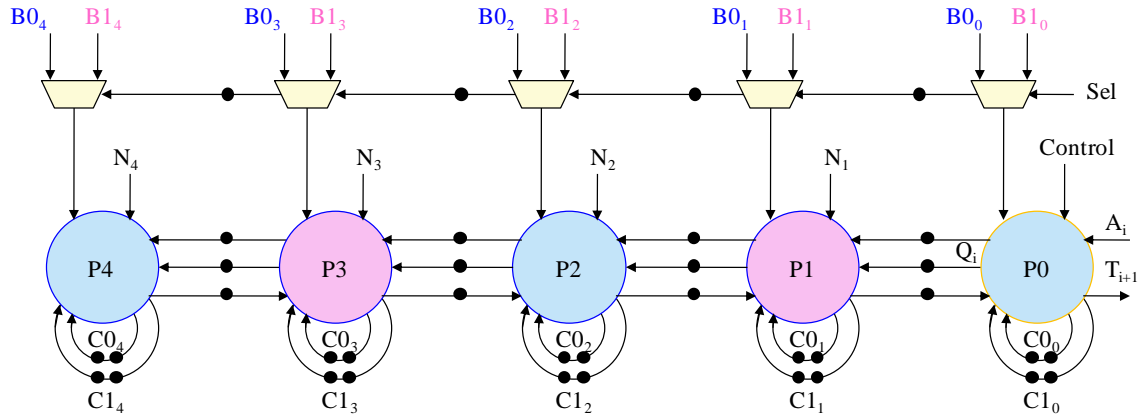


Figura 3.5. Arreglo sistólico para el multiplicador de Montgomery base 2, $n = 4$ bits.

En este arreglo, durante un ciclo de reloj procesan los elementos de procesamiento, P_j , de índice par o impar. Por lo tanto, se usa un multiplexor de 2 a 1 en el dato B , de tal forma que mientras en P_j se procesa $B0$ en P_{j+1} se procesa $B1$, utilizando de esta forma el 100% de los elementos de procesamiento en cada ciclo de reloj. En la Figura 3.5, los P_j de color azul están procesando con el dato $B0$ y los P_j de color rosado están procesando con el dato $B1$. Entonces, al cabo de $4*(n+2)$ ciclos de reloj se obtienen dos productos Montgomery, descritos por las ecuaciones 3.1 y 3.2:

$$T_{AB0} = A*B0*R^{-1} \bmod N \quad (3.1)$$

$$T_{AB1} = A*B1*R^{-1} \bmod N \quad (3.2)$$

3.1.5. Arquitectura del multiplicador de Montgomery base 2

Debido a que el arreglo sistólico tiene una entrada serial (a_i) y una salida serial (T_{i+1}), se usan dos registros de desplazamiento: $SIPO_0$ y $SIPO_1$, los cuales llevan a cabo la conversión paralelo – serie del dato de entrada A y la conversión serie – paralelo de las multiplicaciones de Montgomery T_{AB0} ($PROD0$) y T_{AB1} ($PROD1$). El registro $SIPO_0$ tiene entradas paralela/serial y salida paralela. Este registro tiene dos funciones: generar la entrada serial a_i del multiplicador y recibir la salida serial T_{i+1} del multiplicador correspondiente a la multiplicación de

Montgomery T_{AB0} ($PROD0$), y convertir este resultado en una salida paralela. El registro $SIPO_1$ tiene una entrada serial y una salida paralela. Este registro tiene como función recibir la salida serial T_{i+1} del multiplicador correspondiente a la multiplicación de Montgomery T_{AB1} ($PROD1$) y convertir este resultado en una salida paralela.

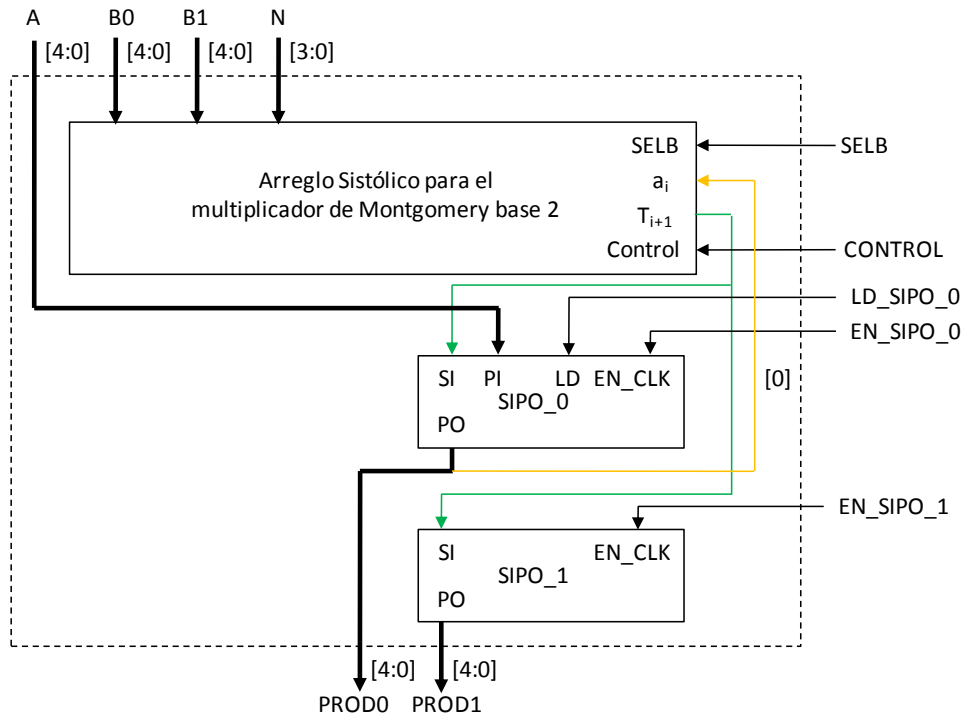


Figura 3.6. *Data path* para el multiplicador de Montgomery base 2, $n = 4$ bits.

En la Figura 3.7 se presenta la máquina de estados algorítmica, “*Algorithmic State Machine*” - ASM, que permite realizar el producto de Montgomery usando el multiplicador sistólico. Esta ASM tiene dos bucles, cuyo número de iteraciones está determinado por el valor inicial del contador $Cont$, el cual es $2(n+2)-1$. Durante el primer bucle, $CTRL = 1$, se calculan los Q_i correspondientes a cada iteración i del Algoritmo 3.2 y los productos parciales de Montgomery usando un sumador *carry-save*, es decir, los bits de suma, T_{i+1} , y dos acarreos, $C0_{i+1}$ y $C1_{i+1}$. En el segundo bucle, $CTRL=0$, se realiza la suma de acarreo propagado de T_i , $C0_i$ y $C1_i$, para obtener el producto de Montgomery, en este caso un bit de $PROD0$ y un bit de $PROD1$ cada ciclo de reloj de forma alternada, y los bits son

almacenados en $SIPO_0$ y $SIPO_1$, respectivamente. La señal FM se activa cuando los dos productos de Montgomery están disponibles y permanece activa hasta que la ASM que controla el multiplicador active la señal Fr . En la Figura 9 se muestra el *controlpath* y el *datapath* para el multiplicador de Montgomery base 2.

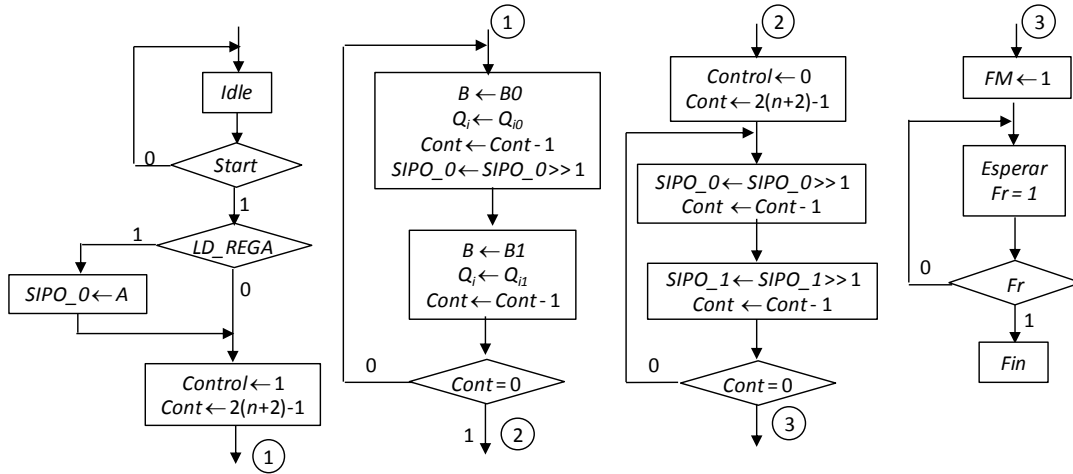


Figura 3.7. Máquina de estados algorítmica para el arreglo sistólico del multiplicador de Montgomery base 2.

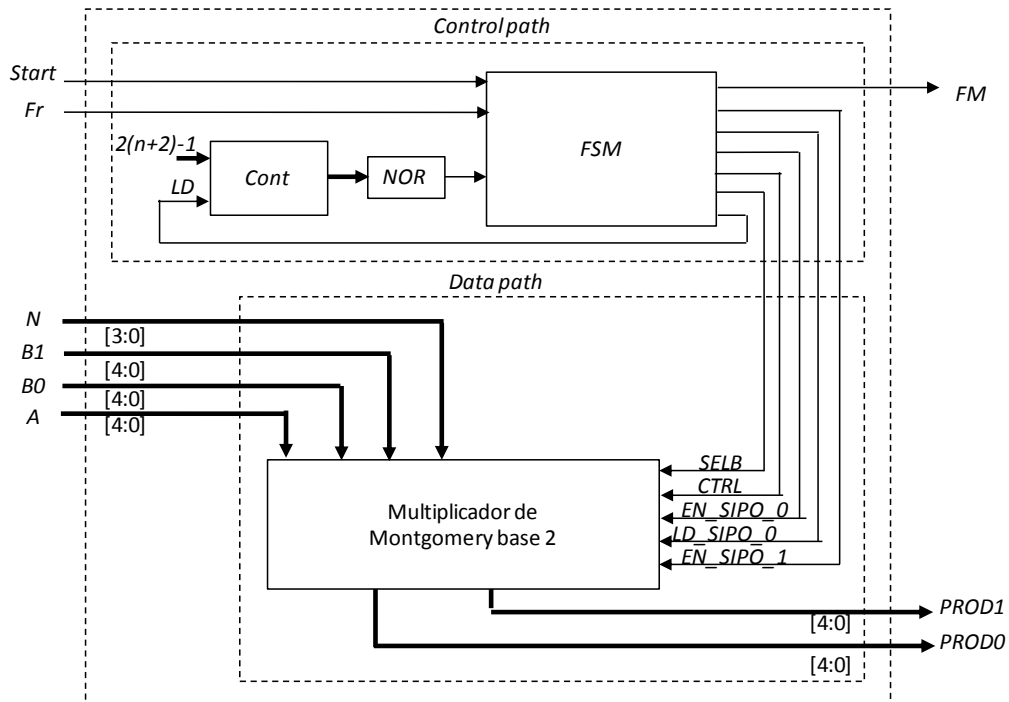


Figura 3.8. Multiplicador de Montgomery base 2: *control path* y *data path*.

3.2. DISEÑO DEL MULTIPLICADOR DE MONTGOMERY BASE 4 USANDO UN ARREGLO SISTÓLICO

En esta sección se presenta el diseño del multiplicador de Montgomery base 4 teniendo en cuenta el procedimiento usado para el diseño del multiplicador de Montgomery base 2.

3.2.1. Algoritmo de Montgomery base 4

El algoritmo de Montgomery base 4 se deduce a partir del Algoritmo 2.2. En este caso, primero se presenta la notación usada y luego en el Algoritmo 3.3 se describe el algoritmo de Montgomery base 4.

La notación usada es la siguiente:

- Los datos A , B y N son representados en base 4, así:

$$A = \sum_{i=0}^n a_i 4^i \quad (3.3)$$

donde $0 \leq a_i \leq 3$.

- N es el módulo, $N = (N_0, \dots, N_{n-1})_4$.
- A es un entero representado como un vector, cuyos componentes son los dígitos base 4 de A , es decir, $A = (a_0, \dots, a_n)_4$.

N' se obtiene usando el Algoritmo 2.3.

Algoritmo 3.3: Multiplicación Modular de Montgomery

base 4

Entradas: $A = (a_0, \dots, a_n)_4 \leq 2N - 1$ con $a_n \leq 1$,

$$B = (b_0, \dots, b_n)_4 \leq 2N - 1$$

Precálculo: $N' = -N^{-1} \bmod 4$ **Variable temporal:** $T = (t_0, \dots, t_n)_4 \leq 2N - 1$ **Salida:** $T = A * B * R^{-1} \bmod N$, $R = r^{n+1}$ y $T \leq 2N - 1$

1. $T := 0$
 2. **for** $i=0$ **to** n **do**
 3. $Q := [t_0 + a_i b_0] * N' \bmod 4$;
 4. $T := [T + a_i B + QN] \text{ div } 4$;
 5. **end for**
 6. **Return** (T)
-

3.2.2. Algoritmo de Montgomery base 4 a nivel de bit

Para llevar a cabo una implementación hardware del algoritmo de Montgomery base 4 usando una arquitectura paralela con buen desempeño es necesario describirlo a nivel de bit y usar sumadores *carry-save*, tal como se presenta en el Algoritmo 3.4, donde nb es la cantidad de bits del módulo N , $a_i B_{1:0}$ son los dos bits menos significativos del producto $a_i * B$ (a_i es un dígito base 4 de A), $a_i B_{4j+k}$ es el bit $4j+k$ del producto $a_i * B$ y $Q_i N_{4j+k}$ es el bit $4j+k$ del producto $Q_i * N$ (Q_i es el cociente correspondiente a la iteración i).

Algoritmo 3.4: Multiplicación de Montgomery base 4 a nivel de bit (sumadores *carry-save*)

Entradas: $A = (a_0, \dots, a_n)_4 \leq 2N - 1$ con $a_n \leq 1$, $B = (B_0, \dots, B_{nb})_2 \leq 2N - 1$, $N = (N_0, \dots, N_{nb-1})_2$

Precálculo: $N' = -N^1 \text{ mod } 4$

Salida: $T = A * B * R^{-1} \text{ mod } N$, $R = r^{n+1}$ y $T \leq 2N - 1$

1. **for** $i = 0$ **to** n **do**
 2. $C_1_{i+1, 1:0} := (T_{i-1:-2} + 2 * C0_{i-1} + 2 * C1_{i-1} + C_1_{i, 1:0}) \text{div} 4;$
 3. **for** $j = 0$ **to** $nb/4$ **do**
 4. **if** $j = 0$
 5. $t1 := (T_{i, 1:-2} + 2 * C0_{i, 1:-1} + 2 * C1_{i, 1:-1} + C_1_{i, 1:0} + 4 * a_i B_{1:0}) \text{div} 4;$
 6. $Q_i := t1 * N' \text{ mod } 4;$
 7. **end if**
 8. **for** $k=0$ **to** 3 **do**
 9. $t := (T_{i, 4j+k} + C0_{i, 4j+k} + C1_{i, 4j+k}) \text{ mod } 2;$
 10. $C0_{i+1, 4j+k-1} := (T_{i, 4j+k} + C0_{i, 4j+k} + C1_{i, 4j+k}) \text{ div } 2;$
 11. $T_{i+1, 4j+k-2} := (t + a_i B_{4j+k} + Q_i N_{4j+k}) \text{ mod } 2;$
 12. $C1_{i+1, 4j+k-1} := (t + a_i B_{4j+k} + Q_i N_{4j+k}) \text{ div } 2;$
 13. **end for**
 14. **end for**
 15. **end for**
 16. $T := T_{n+1} + C0_{n+1} + C1_{n+1}$
-

3.2.3. ARREGLO 2D PARA EL MULTIPLICADOR DE MONTGOMERY BASE 4

La implementación paralela del Algoritmo 3.4 permite obtener un arreglo 2D, donde cada iteración del bucle j representa un elemento de procesamiento, PE_j , y cada iteración del bucle i representa un producto parcial de Montgomery, T_{i+1} , $C0_{i+1}$ y $C1_{i+1}$.

Cada iteración del bucle k realiza la suma de $T_{i,4j+k}$, $CO_{i,4j+k}$, $C1_{i,4j+k}$, $a_i B_{4j+k}$ y $Q_i N_{4j+k}$, usando un sumador *carry-save* de cinco entradas y tres salidas, CSA53, generando dos acarrees $CO_{i+1,4j+k-1}$ y $C1_{i+1,4j+k-1}$, y el bit de la suma $T_{i+1,4j+k-2}$. El sumador CSA53 se muestra en la Figura 3.9. Adicionalmente, es importante notar que el bucle k realiza cuatro iteraciones, por lo tanto, el sumador CSA53 es de cuatro bits, el cual se muestra en la Figura 3.10.

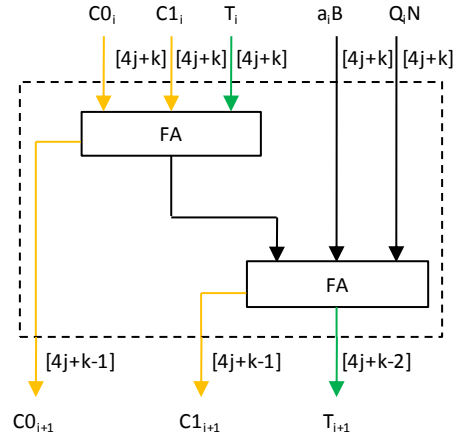


Figura 3.9. Sumador *carry-save* de cinco entradas y tres salidas, CSA53.

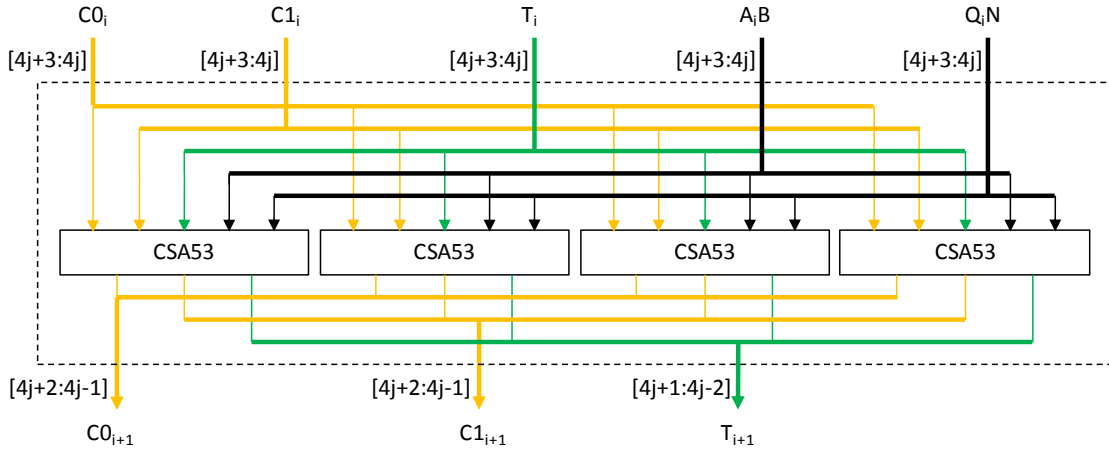


Figura 3.10. Sumador CSA53 de cuatro bits.

De otro lado, cada iteración del bucle j calcula la suma de $T_{i,4j+3:4j}$, $CO_{i,4j+3:4j}$, $C1_{i,4j+3:4j}$, $a_i B_{4j+3:4j}$ y $Q_i N_{4j+3:4j}$, usando el sumador CSA53 de 4 bits presentado en la Figura 3.10. En este caso, se necesitan dos elementos de procesamiento para

realizar el cálculo anterior: $PE0$, para $j = 0$, y PEj , para $j \neq 0$. El elemento de procesamiento $PE0$ calcula Q_i , usando el bloque funcional QS. $PE0$ y PEj son presentados en las Figuras 3.11 y 3.12, respectivamente. En este caso, dos multiplexores 4:1 seleccionan cuatro bits de los productos a_i*B y Q_i*N para cada elemento de procesamiento. El número de elementos de procesamiento es $nb/4 + 1$, donde nb es el número de bits del módulo.

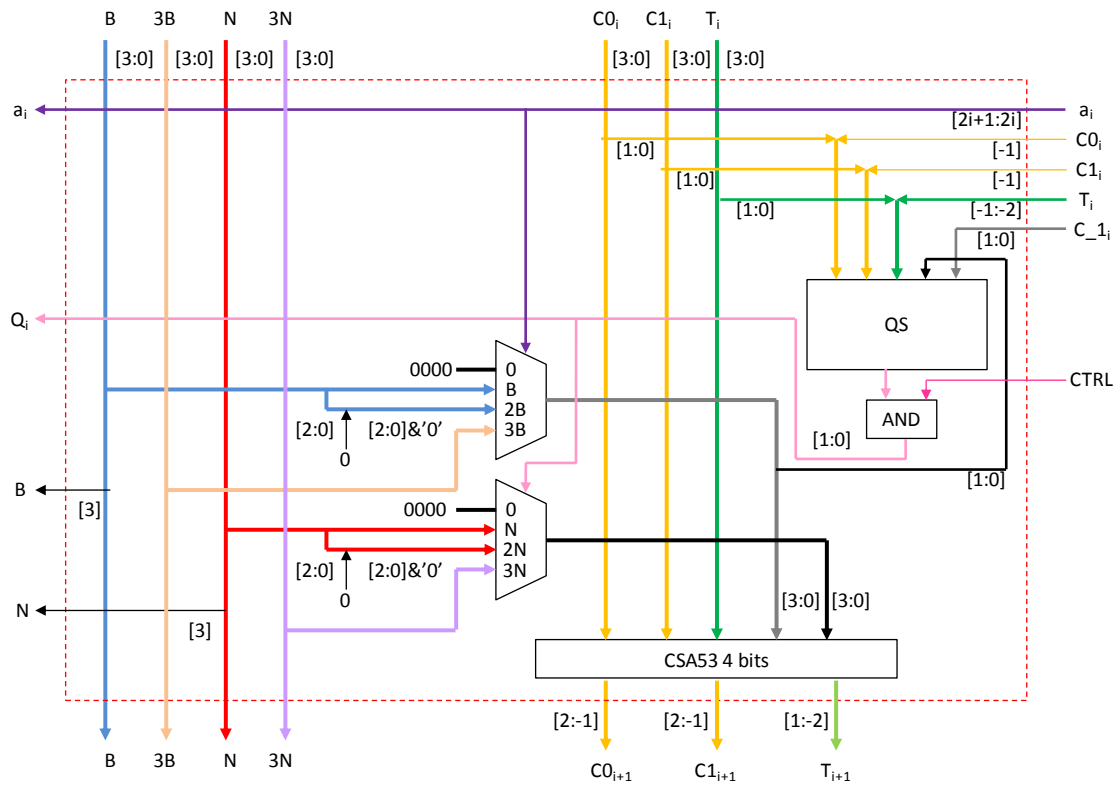


Figura 3.11. Elemento de procesamiento para $j = 0$, $PE0$.

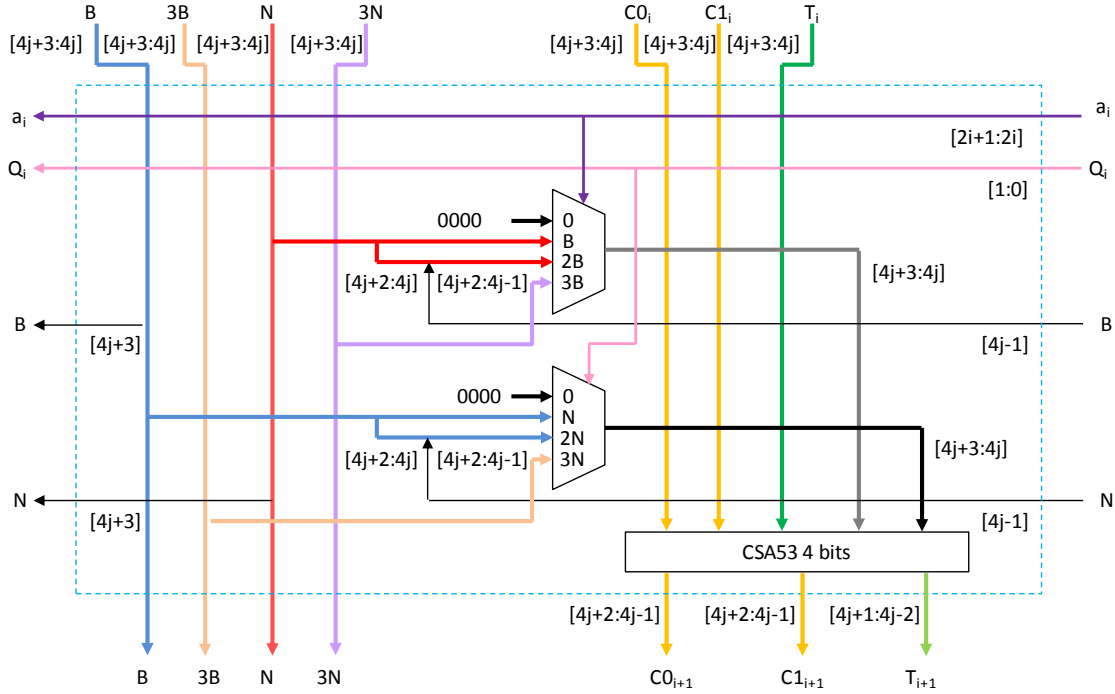


Figura 3.12. Elemento de procesamiento para $j \neq 0$, PE_j .

Como se mencionó anteriormente, el bloque funcional QS calcula Q_i con base en los pasos 5 y 6 del Algoritmo 3.4. El paso 5 lleva a cabo una suma de cinco datos de cuatro bits y una división entre 4. En este caso, la suma se realiza con un arreglo de sumadores *carry-save* y un sumador de acarreo propagado, y la división entre 4 consiste en descartar los dos bits menos significativos de esta suma. De esta forma se obtiene $t1$, y en el paso 6, se multiplica $t1$ por N' . Q_i es igual a los dos bits menos significativos del resultado anterior. N' puede ser 1 ó 3. Si $N' = 1$, entonces Q_i es $t1_{1:0}$ y si $N' = 3$, entonces Q_i es igual a los dos bits menos significativos de la suma de $t1_{1:0}$ y $2*t1_0$. En la Figura 3.13 se muestra el bloque funcional QS para el caso en que $N' = 3$.

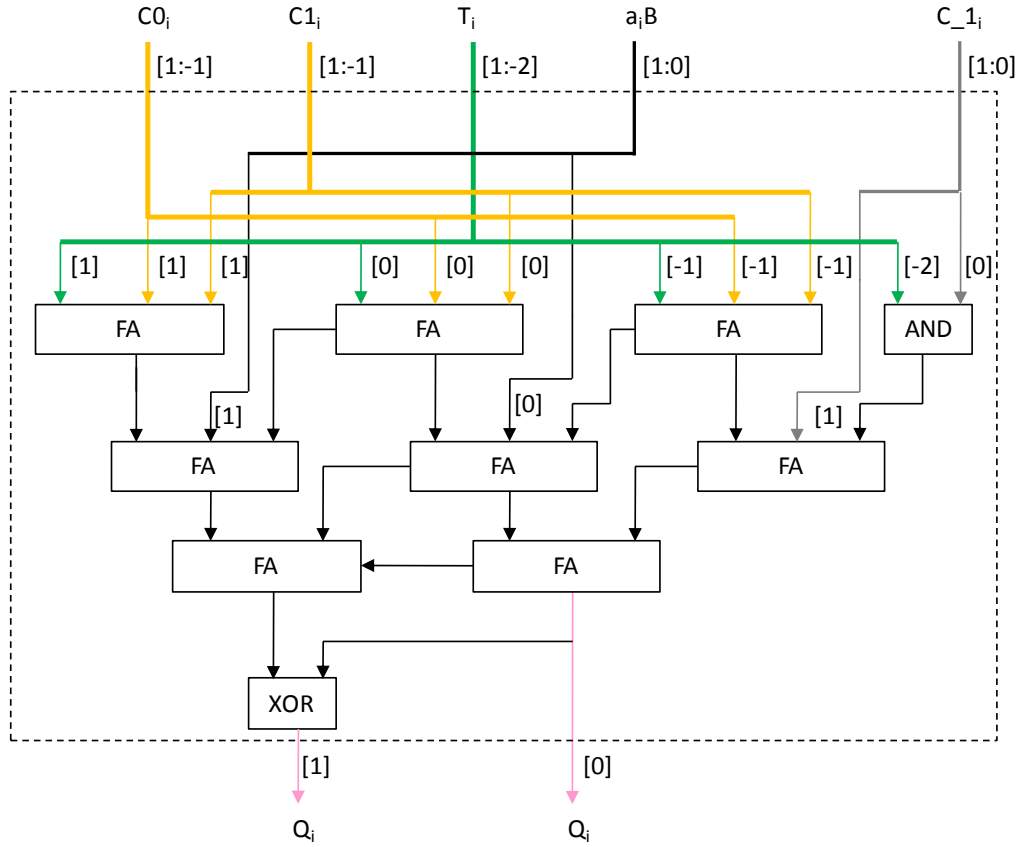


Figura 3.13. Bloque funcional QS para $N' = 3$.

En cada iteración del bucle i se realiza el paso 2. Este paso consiste en el cálculo de los dos acarrees, $C_{i+1,1:0}$, generados por la suma de $T_{i,1:2}$, $C0_{i,1:-1}$, $C1_{i,1:-1}$ y $C_{i,1:0}$. Este cálculo se realiza en el bloque funcional CS_1 . Adicionalmente, este bloque funcional efectúa la suma de T_{n+1} , $C0_{n+1}$ y $C1_{n+1}$ usando un sumador de acarreo de propagado. El bloque funcional CS_1 es presentado en la Figura 3.14.

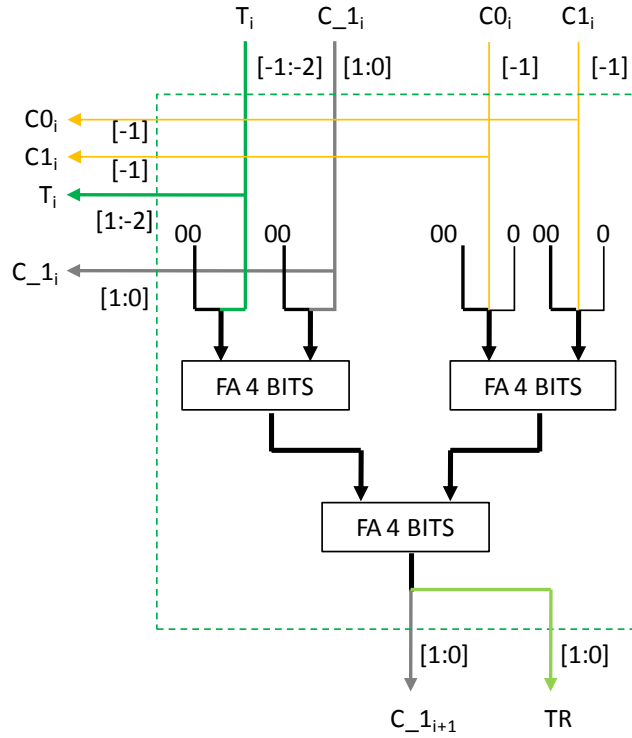


Figura 3.14. Bloque funcional CS_1.

En el Algoritmo 3.4, cada iteración del bucle i genera un arreglo 1D, el cual es formado por el bloque funcional CS_1 y los elementos de procesamiento PE0 y PEj. La Figura 3.15 muestra el arreglo 1D de la iteración i para $n = 4$ dígitos base 4 y $nb = 8$ bits. Los acarrees $C0_i$, $C0_{i+1}$, $C1_i$ y $C1_{i+1}$ se representan con C_i y C_{i+1} .

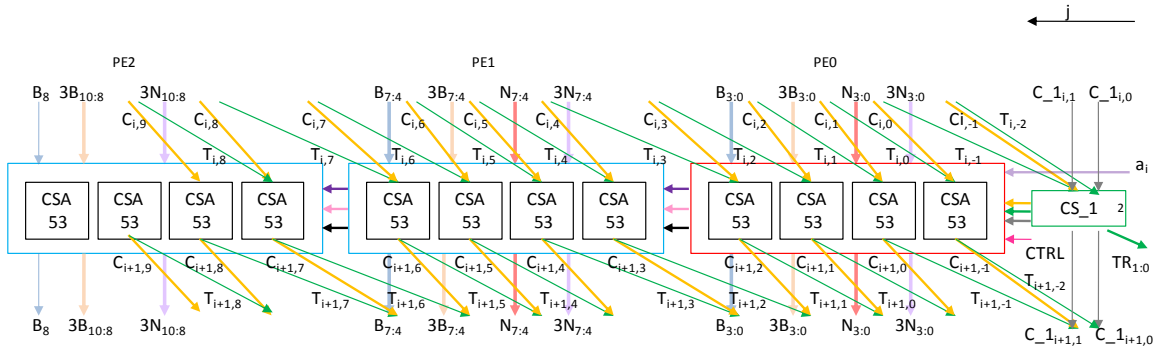


Figura 3.15. Arreglo 1D para la iteración i , $n = 4$ dígitos base 4 y $nb = 8$ bits.

Con base en el Algoritmo 3.4 y usando los elementos de procesamiento $PE0$, PEj y el bloque funcional CS_1 se diseña la arquitectura paralela o 2D para el multiplicador de Montgomery Base 4, el cual se presenta en la Figura 3.16, para $n=4$ dígitos base 4 ó $nb=8$ bits (para el módulo). Cada elemento de procesamiento $PE0$ y PEj se tiene cuatro sumadores $CSA53$, y desde estos sumadores es posible observar la división entre 4. En esta Figura, la esquina superior derecha de cada elemento de procesamiento tiene un número que indica el instante en que procesa cada elemento. Esta secuencia de procesamiento es fundamental para realizar el mapeo del arreglo sistólico.

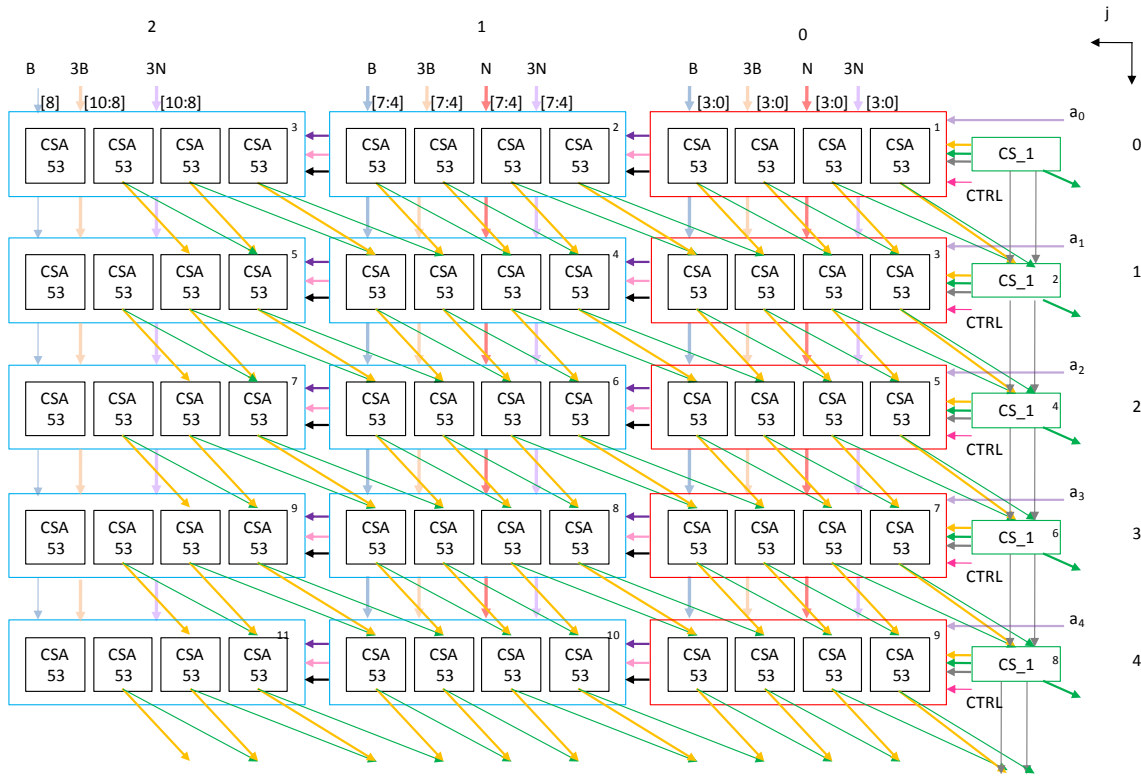


Figura 3.16. Arreglo 2D para el algoritmo de Montgomery base 4 a nivel de bit, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).

3.2.4. Arreglo sistólico para el multiplicador de Montgomery Base 4

Con base en el arreglo 2D de la Figura 3.16 se realiza el mapeo del arreglo sistólico presentado en la Figura 3.17, dónde A y B son los datos a multiplicar y N

es el módulo de $nb=8$ bits. El arreglo sistólico realiza el producto modular de Montgomery en dos procesos: durante los primeros $2(n+1)$ ciclos la señal $CTRL$ es '1', en los cuales se obtiene el Q_i correspondiente a cada dígito base 4 de A , y en los últimos $2(n+1)$ ciclos la señal $CTRL$ es '0', en los que se obtienen dos bits de la multiplicación de Montgomery cada dos ciclos de reloj, desde el bit menos significativo hasta el bit más significativo a través de la salida TR del bloque funcional CS_1 .

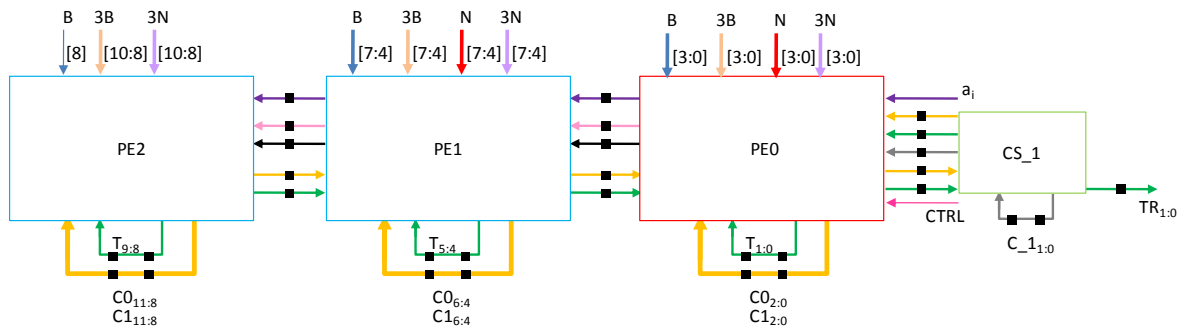


Figura 3.17. Arreglo sistólico para el multiplicador de Montgomery base 4, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).

El procesamiento de cada elemento de procesamiento de la Figura 3.16 se realiza cada dos ciclos de reloj. Por lo tanto, para que el arreglo sistólico procese en cada ciclo de reloj, se pueden multiplexar dos datos para generar B : $B0$ y $B1$, tal como se realizó para el multiplicador de Montgomery Base 2. El multiplicador sistólico con esta modificación se presenta en la Figura 3.18. Este multiplicador calcula simultáneamente dos productos de Montgomery según por las ecuaciones 3.1 y 3.2. Esto permitirá llevar a cabo de forma paralela el cuadrado y el producto de la exponenciación binaria. El cálculo de $3B0$ y $3B1$ se lleva a cabo usando el bloque multiplicador por 3, $Multx3$, presentado en la Figura 3.19, el cual realiza la suma del dato y su múltiplo de 2.

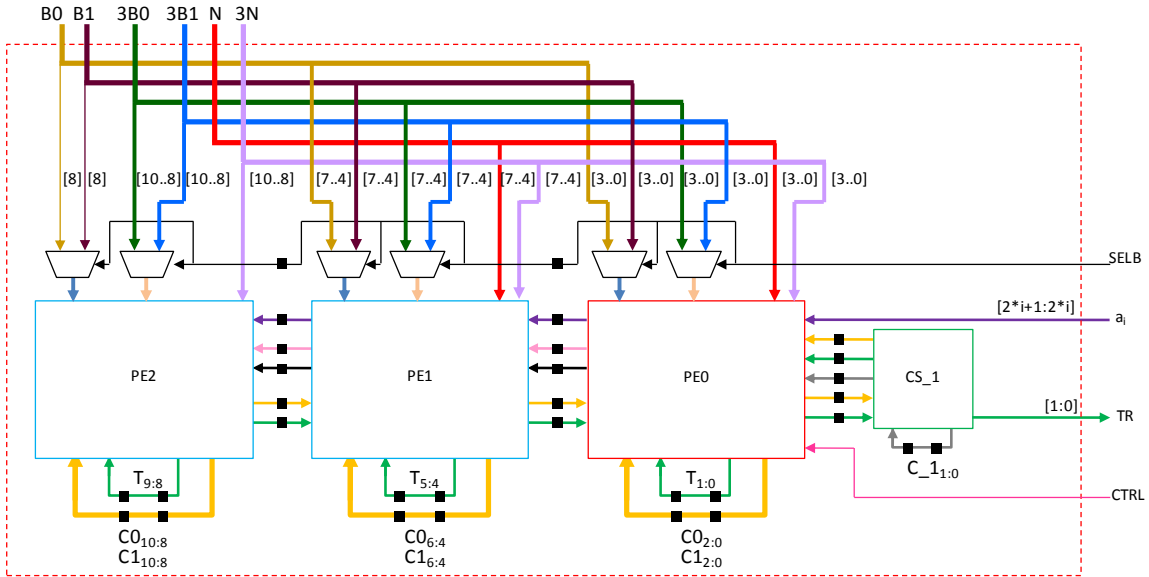


Figura 3.18. Arreglo sistólico del multiplicador de Montgomery base 4 que realiza los productos $AxB0$ y $AxB1$.

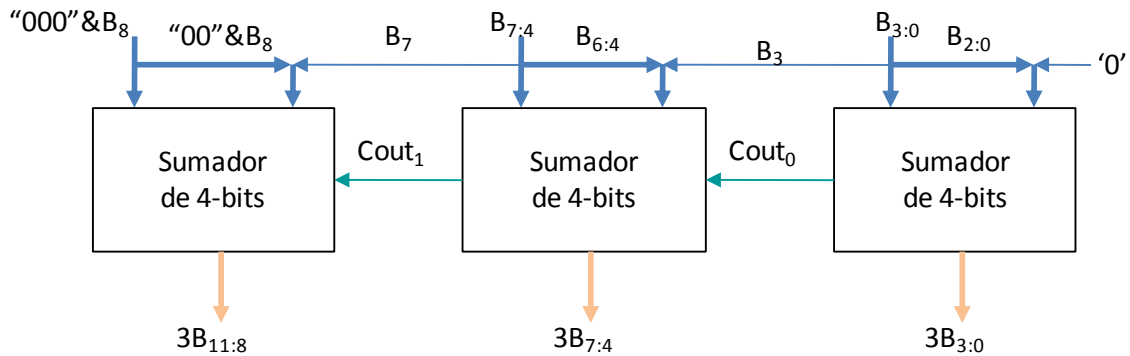


Figura 3.19. Multiplicador por 3, $Multx3$, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).

3.2.5. Arquitectura del multiplicador de Montgomery base 4

Debido a que el arreglo sistólico tiene una entrada de dos bits (a_i) y una salida dos bits (TR), se usan dos registros de desplazamiento: $SIPO_0$ y $SIPO_1$, cada uno de los cuales tiene dos registros SIPO, para los bits de posición par y para los bits de posición impar. Estos registros de desplazamiento llevan a cabo las conversiones entrada paralela – salida dígito base 4 (para la entrada a_i) y entrada

dígito base 4 - salida paralela (para las salidas *PROD0* y *PROD1*). El registro *SIPO_0* tiene entrada paralela, entrada dígito base 4 y salida paralela. Este registro tiene dos funciones: generar la entrada a_i de dos bits del multiplicador y recibir la salida *TR* de dos bits del multiplicador correspondientes al producto *PROD0*, y convertir este resultado a una forma paralela. El registro *SIPO_1* tiene una entrada de dos bits y salida paralela. Este registro tiene como función recibir la salida *TR* de dos bits del multiplicador correspondiente al producto *PROD1* y convertir este resultado a una forma paralela.

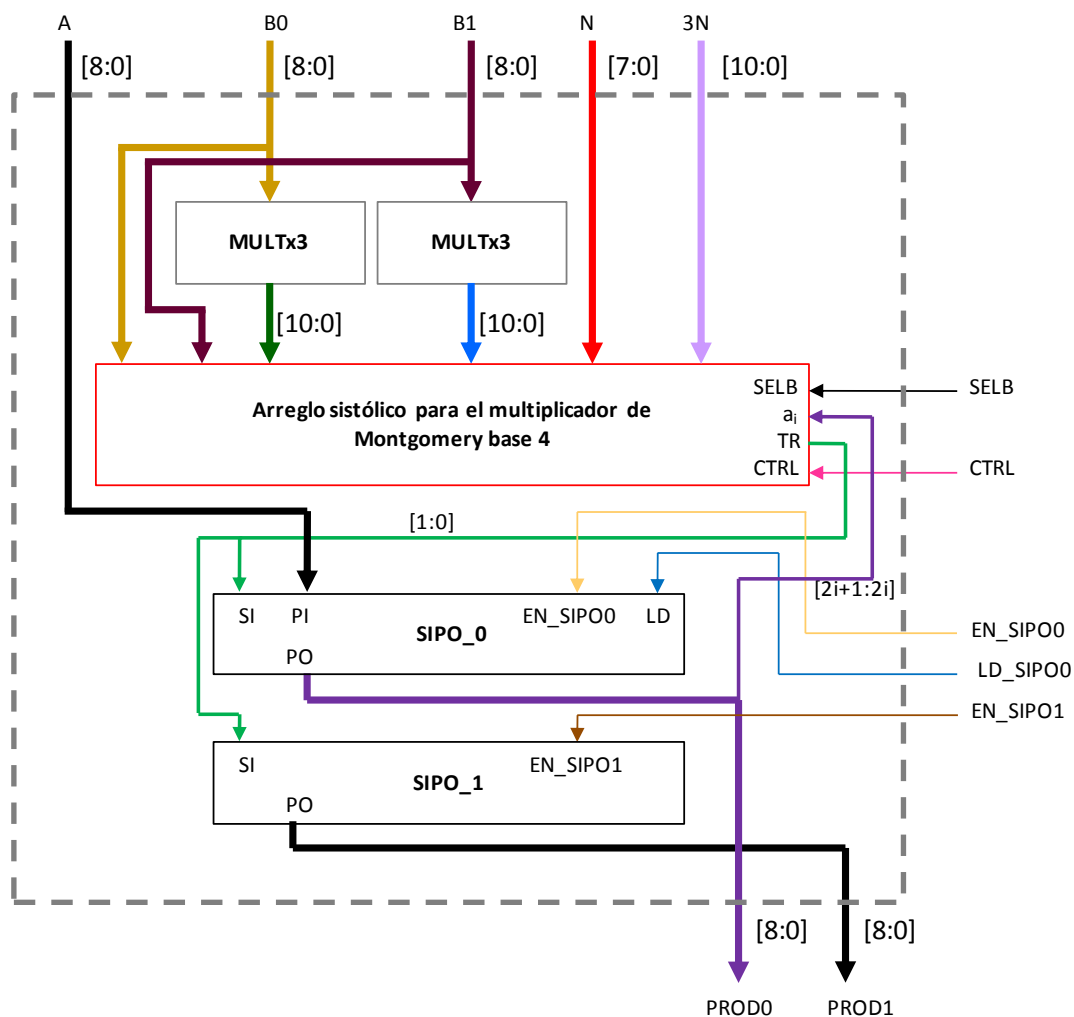


Figura 3.20. Multiplicador modular de Montgomery Base 4, $n = 4$ dígitos base 4 ó $nb = 8$ bits (para el módulo).

En la Figura 3.21 se presenta la máquina de estados algorítmica, “*Algorithmic State Machine*” – *ASM*, que lleva a cabo la multiplicación de Montgomery usando el arreglo sistólico. Esta *ASM* inicia la operación de la multiplicación usando dos estados para calcular los 8 bits menos significativos de $3B0$ y $3B1$, usando el bloque *Multx3* de la Figura 3.19. Los bits restantes de estos precálculos se calculan en los siguientes ciclos desde los menos significativos hasta los más significativos, de forma paralela con los siguientes estados de la máquina. La *ASM* tiene dos bucles, cuya cantidad de ciclos está determinada por el valor inicial del contador *Cont*, el cual es $2(n+1)-1$. Durante el primer bucle, $CTRL=1$, se calculan los Q_i correspondientes a cada iteración del Algoritmo 3.3 junto con los productos parciales obtenidos desde los sumadores *carry-save*. En el segundo bucle, $CTRL=0$, se realiza la suma de acarreo propagado de T_{n+1} , $C0_{n+1}$ y $C1_{n+1}$, obteniendo dos bits de *PROD0* (producto de Montgomery $AxB0$) y dos bits de *PROD1* (Producto de Montgomery $AxB1$) durante cada ciclo de reloj de forma alternada, desde los dos bits menos significativos hasta los dos bits más significativos, y estos dos bits son almacenados en *SIPO_0* y *SIPO_1*, respectivamente. La señal *FM* se activa después de que los dos productos de Montgomery están disponibles y permanece en este estado hasta que la señal *Fr* se active. En la Figura 3.22 se muestra el *control path* y el *data path* para el multiplicador de Montgomery base 4.

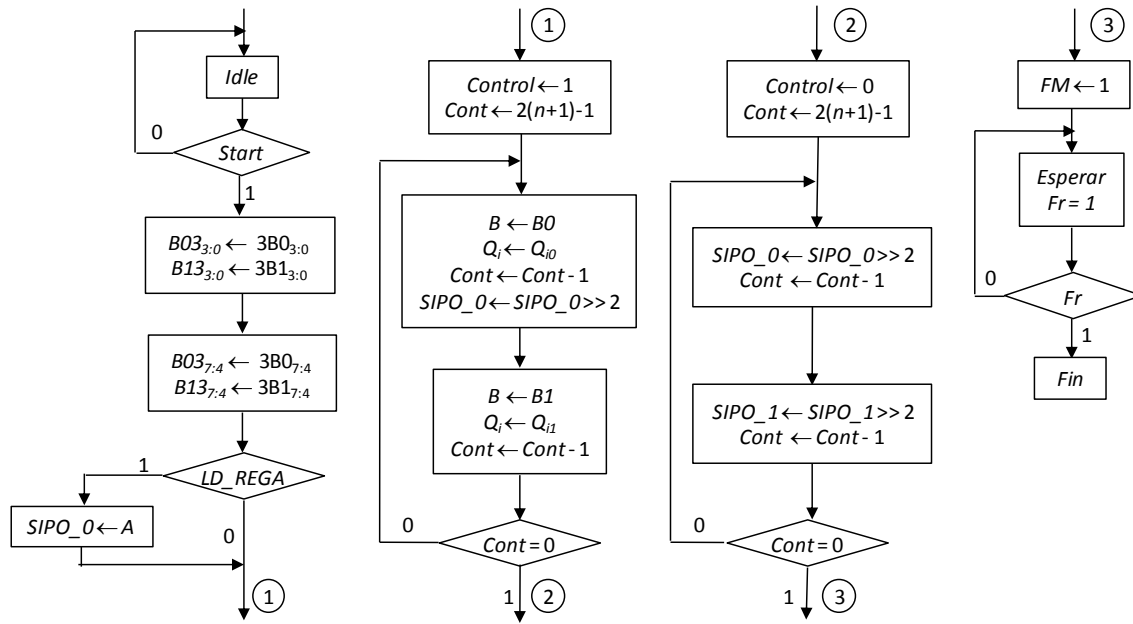


Figura 3.21. Máquina de estados algorítmica del multiplicador de Montgomery Base 4.

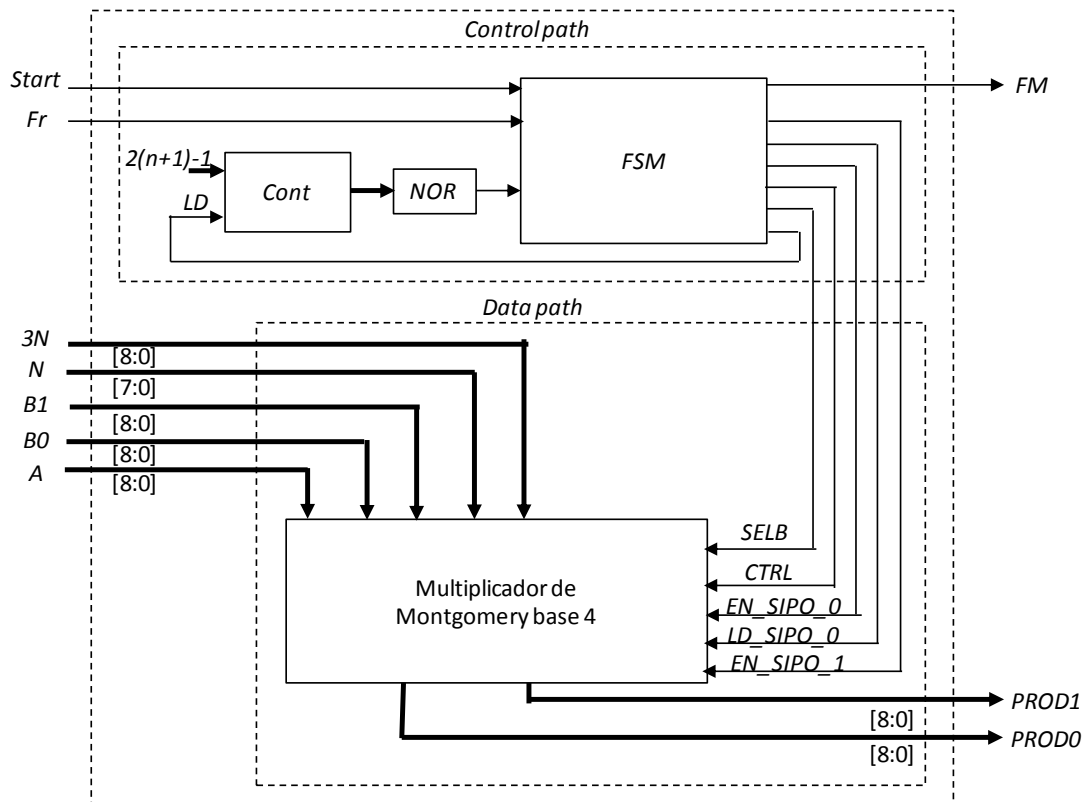


Figura 3.22. Multiplicador de Montgomery base 4, $nb = 8$ bits.

3.3. SELECCIÓN DEL ALGORITMO PARA LA EXPONENCIACIÓN MODULAR

En este trabajo, el algoritmo para la exponenciación modular es seleccionado teniendo en cuenta el mejor compromiso desempeño-área. El Algoritmo 2.6, conocido como exponenciación binaria *LSB-first*, consiste en realizar un bucle de k iteraciones; cada iteración realiza uno ó dos productos de Montgomery, es decir, el cuadrado y el producto modular. Los dos multiplicadores diseñados pueden realizar los dos productos en forma paralela. Por lo tanto, el tiempo total de cálculo de la exponenciación modular es $T \approx k * T_{MM}$, donde k es el número de bits del exponente y T_{MM} es el tiempo de ejecución de los dos productos Montgomery realizados en paralelo. El Algoritmo 2.5 (exponenciación binaria *MSB-first*) también realiza un bucle de k iteraciones, en cada una de las cuales se ejecuta uno o dos productos Montgomery, es decir, el producto modular y el cuadrado, pero en este caso, estos dos productos no pueden ser realizados en forma paralela por los dos multiplicadores diseñados. Los Algoritmos 2.6 (exponenciación modular m-aria) y 2.7 (exponenciación modular con el método de ventana deslizante) realizan la exponenciación modular con menos productos que el Algoritmo 2.5, y en cada iteración se realiza una exponenciación modular y un producto modular. La Tabla 3.1, presentada en [2], muestra el promedio de multiplicaciones realizadas por los algoritmos de exponenciación modular: binaria *LSB-first*, binaria *MSB-first*, m-aria y ventana deslizante, donde d es el tamaño de la ventana para los dos últimos casos.

TABLA 3.1: PROMEDIO DE MULTIPLICACIONES PARA DIFERENTES ALGORITMOS DE EXPONENCIACIÓN MODULAR

k	<i>LSB</i>	<i>MSB</i>	m-aria		ventana-deslizante	
			d	P. Mult.	d	P. Mult.
512	768	512	5	635	5	607
1024	1536	1024	5	1246	6	1195
2048	3072	2048	6	2439	7	2360

En este caso, durante el procesamiento del algoritmo de la exponenciación binaria se puede realizar el cuadrado y el producto modular de cada iteración en forma paralela usando los dos multiplicadores diseñados, por lo tanto el tiempo de ejecución para el caso de $k=2048$ bits es $T \approx 2048 * T_{MM}$. Para los algoritmos de la exponenciación binaria *MSB first*, la exponenciación m-aria y ventana deslizante, el tiempo de ejecución está basado en la cantidad de multiplicaciones dadas en la Tabla 3.1, debido a que no es posible realizar en cada iteración de estos algoritmos el cuadrado o la exponenciación modular, y el producto modular en forma paralela. Por lo tanto, para el caso de $k=2048$ bits, los tiempos promedios de ejecución para la exponenciación binaria *MSB first*, m-aria y ventana deslizante son $T_{MSB} \approx 2048 * T_{MM}$, $T_{Maria} \approx 2439 * T_{MM}$ y $T_{VentD} \approx 2360 * T_{MM}$, respectivamente. Entonces, el algoritmo de la exponenciación binaria *LSB first* realiza la exponenciación modular en el menor tiempo. Adicionalmente, de acuerdo con [2], los algoritmos de exponenciación m-aria y ventana deslizante demandan más recursos que el algoritmo de la exponenciación binaria, haciéndolos más adecuados para implementaciones *software*. Teniendo en cuenta lo anterior y debido a que el criptoprocador RSA tiene un tamaño de clave de 8192 bits, los algoritmos que proporcionan un mejor desempeño para llevar a cabo la exponenciación modular en este diseño son los algoritmos de exponenciación binaria: *LSB-first* y *MSB-first*. En este caso, se seleccionó la exponenciación binaria *LSB-first* debido a que su procesamiento usa menos ciclos usando los multiplicadores de Montgomery diseñados.

3.4. DISEÑO DEL CRIPTOPROCESADOR RSA

El diseño del criptoprocador consiste de un *data path* que permite realizar la multiplicación modular y de una unidad de control que implementa el Algoritmo 2.6 (exponenciación binaria *LSB first*), como se muestra en la Figura 3.23. Es importante anotar que el *data-path* y el *control-path* son iguales para los criptoprocadores RSA que usan los multiplicadores de Montgomery base 2 y base 4.

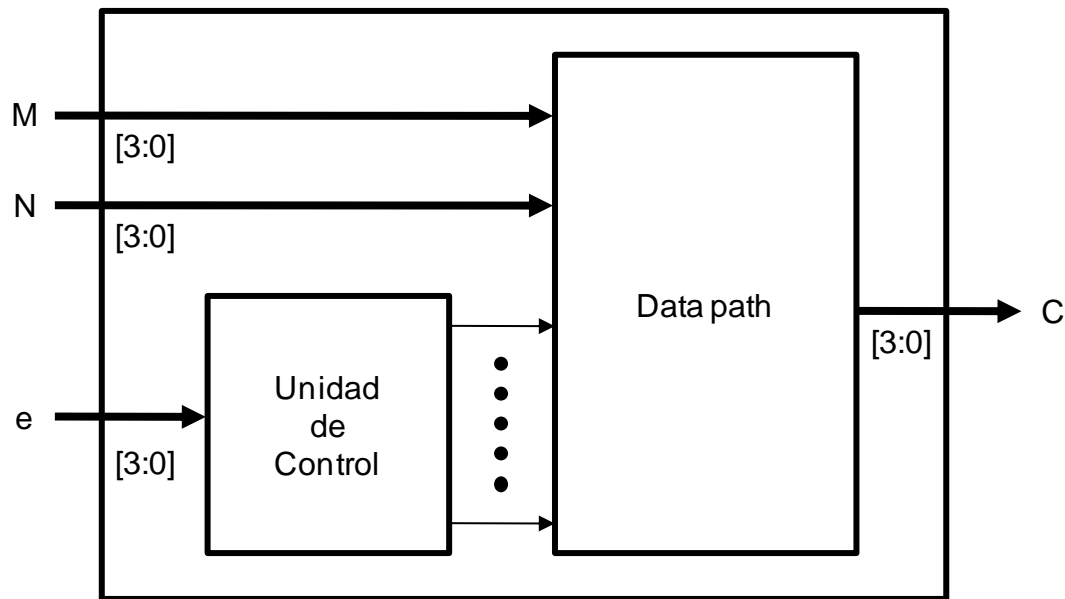


Figura 3.23. Diagrama de bloques del Criptoprocador RSA

3.4.1. Data-path del Criptoprocador RSA

En la Figura 3.27 se muestra el *data path* del criptoprocador *RSA* usando el multiplicador de Montgomery base 4, para $n = 4$ dígitos base 4 y $nb = 8$ bits (para el módulo). El dato de salida del bloque *SEL_A* es cargado en el registro *SIPO_0* del multiplicador de Montgomery base 4 (ver Figura 3.20) cuando se activa la señal *LD_REGA*. El bloque *SEL_A* es presentado en la Figura 3.28, el cual tiene la siguiente función lógica: cuando $UNO_A = 0$, la salida de *SEL_A* es la constante *R2*, y cuando $UNO_A = 1$, la salida de *SEL_A* es el número 1. El registro *REG_M* determina el dato *B0* del multiplicador. El Multiplexor *MUX* selecciona el dato a cargar en *REGB0*, es decir, si $SELB0 = 0$ y $ENR = 1$, se carga *M* en *REGB0*, y si $SELB0 = 1$ y $ENR = 1$, se carga *PROD0* en *REG_M*. El registro *REG_C* determina el dato *B1* del multiplicador. La activación de UNO_C , realizará un *reset* de los bits 8:1 y un *set* del bit 0 de *REG_C*, es decir, este registro se carga con el número 1. El dato sincrónico que se carga en este registro es *PROD1*.

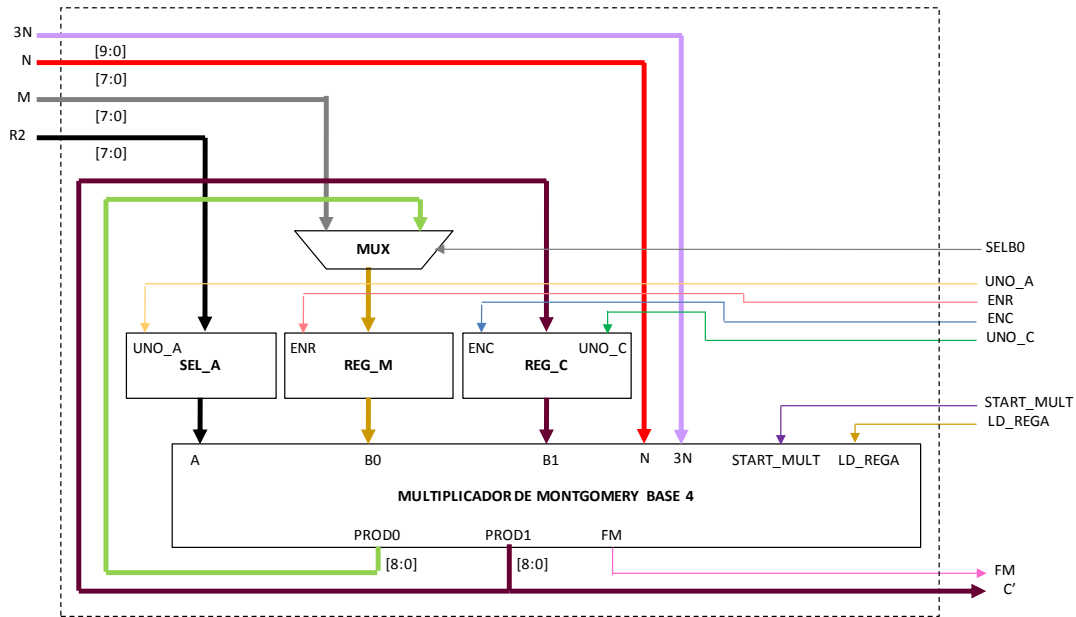


Figura 3.24. *Data path* del criptoprocador *RSA* usando el multiplicador de Montgomery base 4.

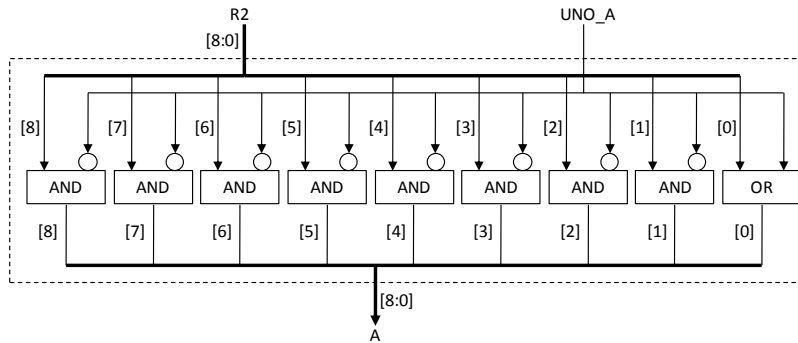


Figura 3.25. Bloque SEL_A .

3.4.2. Unidad de control del Criptoprocador *RSA*

La máquina de estados de la Unidad de control permite realizar la exponenciación binaria *LSB first*, es decir, la *FSM* se encarga de ejecutar el Algoritmo 2.6 usando el multiplicador de Montgomery base 2 ó base 4. En la Figura 3.29 se muestra la máquina de estados algorítmica para realizar la exponenciación modular. Los primeros tres estados realizan la inicialización de M' y C' , calculando los productos de Montgomery de $R2$ y M , y de $R2$ y 1, respectivamente. En este caso, $SIPO_0$

es cargado con la constante $R2$, REG_M es cargado con M y REG_C es cargado con el número 1. Estos dos productos se efectúan de forma paralela en el multiplicador de Montgomery base 4. Durante la realización de la exponenciación binaria, M' es almacenado en $SIPO_0$ y REG_M , y C' es almacenado en REG_C . Luego, se realiza un bucle de k iteraciones que consiste en determinar C' calculando el producto de Montgomery de M' y C' , si el bit del exponente correspondiente a la iteración es 1, y determinar M' calculando el producto de Montgomery de M' y M' . Después el registro $SIPO_0$ es cargado con el número 1, usando el bloque SEL_A y la señal LD_SIPO0 , y se efectúa el producto de Montgomery de C' y el número 1, para calcular la exponenciación modular. En la Figura 3.30 se muestra el *control path* y el *data path* del criptoprocador RSA usando el multiplicador de Montgomery base 4, para $n = 4$ dígitos base 4 y $nb = 8$ bits (para el módulo).

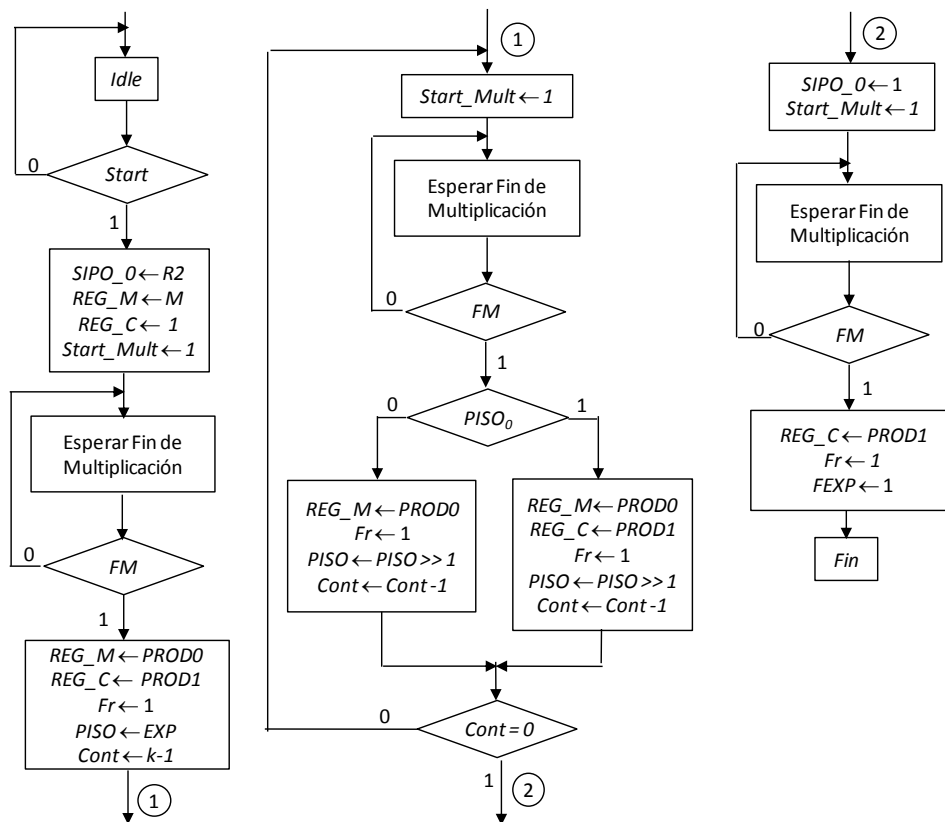


Figura 3.26. ASM de la máquina de estados del criptoprocador RSA usando el multiplicador de Montgomery base 2 ó base 4.

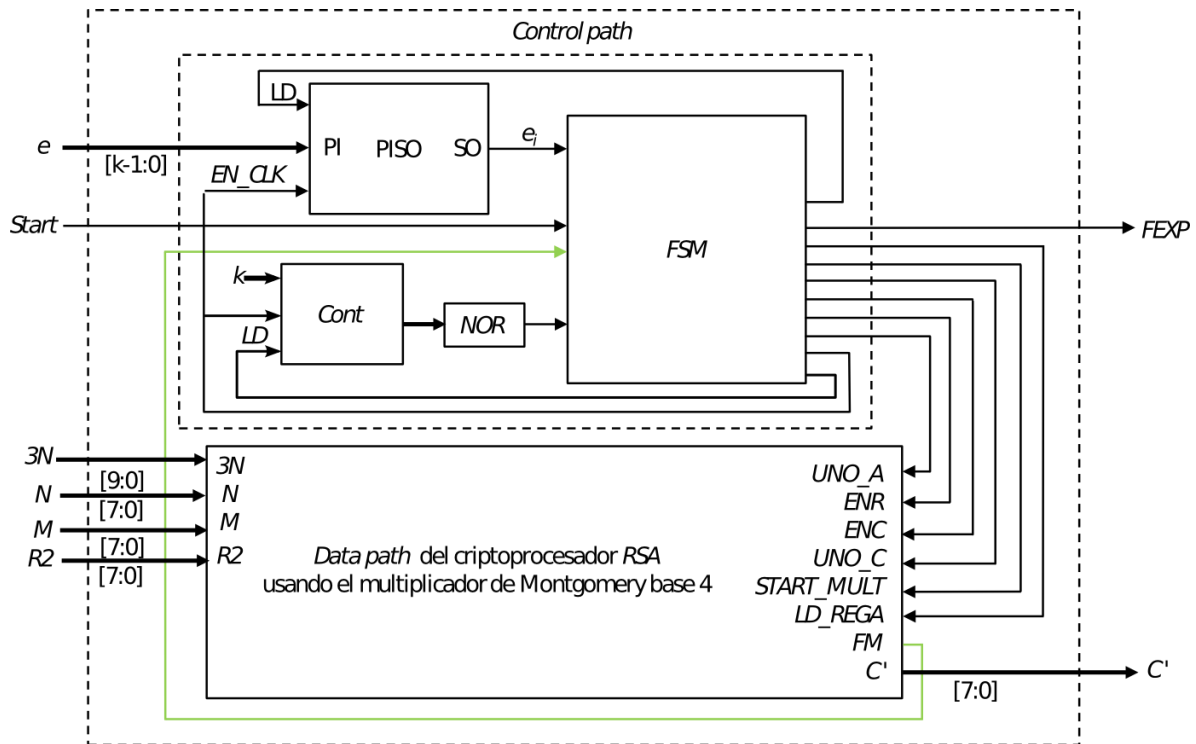


Figura 3.27. *Data path* y *control path* del criptoprocador RSA usando el multiplicador de Montgomery base 4, $nb = 8$ bits.

CAPÍTULO 4

RESULTADOS DE VERIFICACIÓN Y SÍNTESIS PARA LOS MULTIPLICADORES DE MONTGOMERY Y LOS CRIPTOPROCESADORES RSA

Este capítulo presenta los resultados de verificación en *hardware* y síntesis de los multiplicadores de Montgomery base 2 y 4, y dos Criptoprocesadores RSA diseñados con los multiplicadores anteriores. Los resultados de verificación en *hardware* son comparados con los resultados de simulación obtenidos con Maple.

La organización de este capítulo es como sigue. La Sección 4.1 presenta la simulación funcional en Maple de la multiplicación de Montgomery y la exponenciación binaria. La Sección 4.2 presenta la verificación en *hardware* de los criptoprocesadores RSA usando los vectores de RSA Labs para una clave de 1024 bits y 2048 bits; y usando vectores propios para una clave de 8192 bits. La Sección 4.3 presenta los resultados de síntesis de los multiplicadores de Montgomery base 2 y 4, y los criptoprocesadores RSA.

4.1. SIMULACIÓN FUNCIONAL DE LOS ALGORITMOS DE MONTGOMERY Y EXPONENCIACIÓN BINARIA

En esta sección se presenta la simulación funcional, usando Maple, del algoritmo de Euclides extendido, la Ecuación 2.9 para el producto de Montgomery, los algoritmos de Montgomery base 2 y base 4, y el algoritmo de la exponenciación binaria.

4.1.1. Simulación del algoritmo de Euclides extendido

El Algoritmo 2.1 se simuló en Maple usando el procedimiento *algeuext*, el cual se muestra en la Figura 4.1. Este procedimiento lleva a cabo dos cálculos: $R^{-1} \bmod N$ ($R1$) y $-N^{-1} \bmod R$ (NP). El inverso multiplicativo de $R \bmod N$ ($R1$) se usa para calcular el producto de Montgomery usando la Ecuación 2.9 y el inverso aditivo del inverso multiplicativo de $N \bmod R$ (NP) se usa en el algoritmo de la multiplicación de Montgomery base 4.

```
algeuext := proc (R, N)

a := R;
b := N;
x := 0;
y := 1;
lastx := 1;
lasty := 0;

while b ≠ 0 do

    quotient := floor(a/b);

    temp := b;
    b := mod(a, b);
    a := temp;

    temp := x;
    x := lastx - quotient*x;
    lastx := temp;

    temp := y;
    y := lasty - quotient*y;
    lasty := temp;

end do;

R1 := N + lastx; #  $R^{-1} \bmod N$ 
NP := -(lasty - R) ; #  $-N^{-1} \bmod R$ 

end proc;
```

Figura 4.1. Procedimiento *algeuext* para simular en Maple el algoritmo de Euclides extendido

4.1.2. Simulación del algoritmo de Montgomery

La Ecuación 2.9 es usada para simular el producto de Montgomery. Esta ecuación calcula el producto modulo N de dos números enteros A , B , y la constante R

$1 \bmod N$, donde $R=2^{n+2}$. La simulación en Maple se realiza con el procedimiento *MMM*, *Montgomery Modular Multiplication*, el cual se muestra en la Figura 4.2. $R^{-1} \bmod N$ se calcula con el procedimiento *algeucext* de la Figura 4.1.

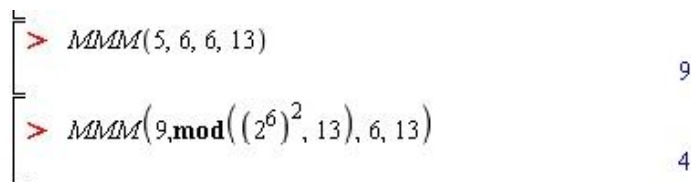
```
MMM := proc(A, B, N, PQ)

    prod := mod(A*B*algeucext(2^N, PQ), PQ);

end proc;
```

Figura 4.2. Procedimiento *MMM* para simular en Maple el producto de Montgomery.

En la Figura 4.3 se presentan los resultados de la prueba realizada con el procedimiento *MMM*, en la cual se obtiene el producto modular de los enteros 5 y 6, módulo 13, usando dos productos de Montgomery, es decir, las Ecuaciones 2.9 y 2.10.



```
> MMM(5, 6, 6, 13)
9
> MMM(9, mod((2^6)^2, 13), 6, 13)
4
```

Figura 4.3. Resultados del procedimiento *MMM*.

Este procedimiento es usado en la verificación de la multiplicación modular usando los algoritmos de Montgomery base 2 y 4.

4.1.3. Simulación del algoritmo de Montgomery base 2

El Algoritmo 3.1, algoritmo de Montgomery base 2, se simuló en Maple usando el procedimiento *hmontmult*, el cual es presentado en la Figura 4.4. En este procedimiento A y B son los números enteros positivos a multiplicar, N es el módulo, nb es el número de bits del módulo más 2 ($n+2$) y k es el logaritmo en base 2 de 2 (base), por lo tanto $k = 1$.

```

hrmontmult := proc (A, B, N, nb,k)

#A,B: numeros a multiplicar
#N: modulo
#nb: número de bits de N + 2
#k: exponente de b.  $r=b^k$ .  $R=r^{(nb+2)}$ 

#Conversion de A a binario

ab := array(1 .. nb);
atemp := A;

for i from 1 by 1 to nb do
    ab[i] := mod(atemp, 2);
    atemp := floor(atemp/2);
end do;

print(ab);

#Conversion de B a binario

bb := array(1 .. nb);
btemp := B;

for i from 1 by 1 to nb do
    bb[i] := mod(btemp, 2);
    btemp := floor(btemp/2);
end do;

print(bb);

#t: Producto de Montgomery
NPON := 1;
t := 0;

for i from 1 by k to nb do
    abk := ab[i];
    qi := (mod(t, 2^k) + abk*(mod(B, 2^k)))*NPON;
    t := floor((t + abk*B + qi*N)/2^k);
    print(t);
end do;

end proc;

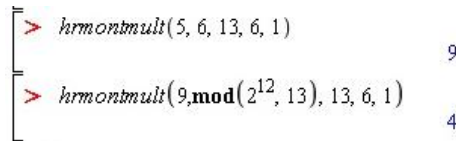
```

Figura 4.4. Procedimiento *hrmontmult* para la simulación en Maple del algoritmo de la multiplicación de Montgomery base 2.

Este procedimiento es dividido en tres etapas:

1. Conversión de A a binario: los bits son almacenados en el vector ab , desde el menos significativo hasta el más significativo.
2. Conversión de B a binario: los bits son almacenados en el vector bb , desde el menos significativo hasta el más significativo.
3. Bucle de nb iteraciones, en el cual se obtienen los cocientes q_i y los productos parciales de Montgomery t .

En la Figura 4.5 se presentan los resultados de la prueba del procedimiento `hmontmult` para la multiplicación de Montgomery base 2, en la cual se obtienen dos productos de Montgomery: el primero multiplica 5 por 6 módulo 13 y el segundo multiplica el resultado anterior por R_2 , para obtener el producto $5 \cdot 6 \bmod 13$, de acuerdo con las Ecuaciones 2.9 y 2.10.



```

> hmontmult(5, 6, 13, 6, 1)
9
> hmontmult(9, mod(2^12, 13), 13, 6, 1)
4

```

Figura 4.5. Resultados del procedimiento para la multiplicación de Montgomery base 2.

4.1.4. Simulación del algoritmo de Montgomery base 4

El Algoritmo 3.3, algoritmo de Montgomery base 4, se simuló en Maple usando el procedimiento `hmontmult`, el cual es presentado en la Figura 4.6. En este procedimiento A y B son los números enteros positivos a multiplicar, N es el módulo, nb es el número de bits del módulo más 2 ($n+2$) y k es el logaritmo en base 2 de 4 (base), por lo tanto $k = 2$. Conversión de B a binario: los bits son almacenados en el vector bb , desde el menos significativo hasta el más significativo.

```

hrmontmult := proc (A, B, N, nb, k)

#A,B: numeros a multiplicar
#N: modulo
#nb: número de bits de N + 2
#k: exponente de b (2).  $r=b^k$ .  $R=r^{nb}$ .  $k=2$ 

#Conversion de A en binario

ab := array(1 .. nb);
atemp := A;

for i from 1 by 1 to nb do
    ab[i] := mod(atemp, 2);
    atemp := floor(atemp/2) ;
end do;

print(ab);

#Conversion de B en binario

bb := array(1 .. nb);
btemp := B;

for i from 1 by 1 to nb do
    bb[i] := mod(btemp, 2);
    btemp := floor(btemp/2) ;
end do;

print(bb);

#t:Producto de Montgomery

exp := nbits/k;
R := (2^k)^exp;
NP := algeuext(R, N); #-N^(-1)modR
NPON := mod(NP, 2^k);
t := 0;

for i from 1 by k to nb do
    abk := 2*ab[i+1]+ab[i];
    qi := mod((mod(t, 2^k) + abk*(mod(B, 2^k)))*NPON, 4);
    t := floor((t + abk*B + qi*N)/2^k);
    print(t)
end do;

end proc;

```

Figura 4.6. Procedimiento *hrmontmult* para simular en Maple el algoritmo de la multiplicación de Montgomery base 4.

Este procedimiento es dividido en cuatro etapas:

1. Conversión de A a binario: los bits son almacenados en el vector ab , desde el menos significativo hasta el más significativo.
2. Conversión de B a binario: los bits son almacenados en el vector bb , desde el menos significativo hasta el más significativo.
3. Cálculo de $-R^{-1} \bmod N$ usando el procedimiento *algeuext* (ver Figura 4.1). Del resultado de este cálculo se usan los dos bits menos significativos, es decir, $NP0N$, para determinar el cociente q_i .
4. Bucle de nb iteraciones, en el cual se obtienen los cocientes q_i y los productos parciales de Montgomery t .

En la Figura 4.7 se presentan los resultados del procedimiento *hmontmult* de la multiplicación de Montgomery base 4, en la cual se obtienen dos productos de Montgomery: el primero multiplica 5 por 6 módulo 13 y el segundo multiplica el resultado anterior por R^2 , para obtener el producto $5 \cdot 6 \bmod 13$, de acuerdo con las Ecuaciones 2.9 y 2.10.

```

> hmontmult(5, 6, 13, 6, 2)
9
> hmontmult(9, mod(4^6, 13), 13, 6, 2)
4

```

Figura 4.7. Resultados del procedimiento para la multiplicación de Montgomery base 4.

4.1.5. Simulación de la exponenciación binaria

En la Figura 4.8 se presenta el procedimiento *exponenciación* para simular en Maple el Algoritmo 2.5, algoritmo de la exponenciación binaria *LSB first*. Este algoritmo usa el procedimiento correspondiente a la multiplicación de Montgomery base 4, y tiene las siguientes entradas: dato (M), número de bits del módulo PQ más dos (N), exponente ($expn$), número de bits del exponente (k), y el módulo (PQ). El procedimiento se divide en tres etapas:

1. Inicialización de M' con el producto de Montgomery de $R^2 \times M$, y de C' con el producto de Montgomery de $R^2 \times 1$.
2. Bucle de k iteraciones, donde se evalúan los bits del exponente $expn$, desde el menos significativo hasta el más significativo. En cada iteración se actualiza M' con el producto de Montgomery de $M' \times M'$ y C' se actualiza con el producto de Montgomery de $M' \times C'$, si el bit del exponente, $expn$, correspondiente a la iteración i es '1'.
3. Se calcula la exponenciación modular, obteniendo el producto de Montgomery de $C' \times 1$.

```
exponenciacion := proc (M, N, expnop, k, PQ)
```

```
  #M: mensaje
```

```
  #N: Numero de bits de PQ + 2
```

```
  #expnop: exponente
```

```
  #k: Numero de bits del exponente
```

```
  #PQ: Modulo
```

```
  R2 := mod(2^N*2^N, PQ);
```

```
  MP := hrmontmult(M, R2, PQ, N, 2);
```

```
  CP := hrmontmult(1, R2, PQ, N, 2);
```

```
  for i from 0 by 1 while i < k do
```

```
    res := mod(expnop, 2);
```

```
    if res = 1 then
```

```
      CP := hrmontmult(MP, CP, PQ, N, 2);
```

```
    end if;
```

```
    MP := hrmontmult(MP, MP, PQ, N, 2);
```

```
    expnop := floor((1/2)*expnop);
```

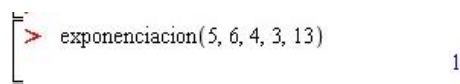
```
  end do;
```

```
  C := hrmontmult(CP, 1, PQ, N, 2);
```

```
end proc;
```

Figura 4.8. Procedimiento *exponenciación* para la simulación en Maple del algoritmo de la exponenciación binaria.

En la Figura 4.9 se muestran los resultados del procedimiento *exponenciación* para la exponenciación binaria de la siguiente operación: $5^4 \bmod 13$.



```

> exponenciacion(5, 6, 4, 3, 13)
1

```

Figura 4.9. Resultados del procedimiento *exponenciación* para la exponenciación binaria.

4.2. VERIFICACIÓN EN *HARDWARE* DE LOS CRIPTOPROCESADORES RSA

En esta sección se presentan los resultados de verificación en *hardware* para comprobar la correcta operación de los criptoprocesadores RSA diseñados. En la primera sub-sección se presentan las verificaciones del Criptoprocesador RSA basado en el multiplicador de Montgomery base 2: usando los vectores de *test* estándar recomendados por RSA Labs para una clave de 1024 bits y usando vectores propios para una clave de 8192 bits. En la segunda sub-sección se presentan las verificaciones del Criptoprocesador RSA basado en el multiplicador de Montgomery base 4: usando los vectores de *test* estándar recomendados por RSA Labs para una clave de 2048 bits y usando vectores propios para una clave de 8192 bits. En este caso, cada resultado de la verificación hardware se efectúa usando SignalTap II ELA [14] y es comparado con los resultados obtenidos usando Maple para el procedimiento *exponenciación* (ver Figura 4.8).

4.2.1. Verificación en *hardware* del Criptoprocesador RSA basado en el multiplicador de Montgomery base 2

En esta sub-sección se presenta la verificación en *hardware* del Criptoprocesador RSA basado en el multiplicador de Montgomery base 2. En este caso se llevan a cabo dos pruebas. La primera prueba lleva a cabo la encriptación y desencriptación de un mensaje de 1024 bits y la segunda prueba lleva a cabo la exponenciación modular de un número de 8192 bits, usando un exponente y un módulo de 8192 bits.

La verificación en *hardware* del Criptoprocador RSA basado en el multiplicador de Montgomery base 2 que lleva a cabo la encriptación y desencriptación de un mensaje de 1024 bits se realiza usando los vectores de *test* estándar recomendados por RSA Labs [15]. Para los dos procesos se presentan los resultados de simulación obtenidos en Maple y la verificación en *hardware* usando SignalTap II ELA.

Para llevar a cabo el proceso de encriptación usando el procedimiento *exponenciación*, primero se efectúa la conversión hexadecimal-decimal del módulo N (vector de *test* de RSA Labs), la cual se muestra en la Figura 4.10. En este caso, el exponente $e = 3$. En la Figura 4.11 se muestran los resultados del procedimiento *exponenciación* para la encriptación y la conversión del resultado a hexadecimal, con el fin de realizar la comparación con los resultados obtenidos de la verificación en *hardware*.

```
> convert(
  "ACD1CC46DFE54FE8F9786672664CA2690D0AD7E5003BC6427954D939EEE8B27152E6A947450D7FA980172DE064D6569A28A83FE70FA840\
  F5E9802CB8984AB34BD5C1E6399EC21E4D3A3A69BE4E676F395AAFEF7C4925FD4FAEE9F9E5E48AF4315DF0EC2DB9AD7A350B3DF2F4D\
  15DC0039846D1ACA3527B1A75049E3FE34F43BD", decimal, hexadecimal)
12135799630782258292480003118273039063451948068513463490667538495641920489459126605197868633220535994712906637614439296855765652994\
65875334911731688425836289995273880779057143489240576953895565259046577277564045340434446742781023347212730698788855269798\
892110420823258571915752889030233244039071524668349
```

Figura 4.10. Resultados de la conversión hexadecimal-decimal del módulo N usando Maple

```
> exponenciacion(2^1022 + 1, 1026, 3, 2,
  1213579963078225829248000311827303906345194806851346349066753849564192048945912660519786863322053599471290663761443929685576565\
  299465875334911731688425836289995273880779057143489240576953895565259046577277564045340434446742781023347212730698788855269798\
  5666892110420823258571915752889030233244039071524668349)
10856162958564428718915055129272308878463414305154012725138925460185863321065114457913026545688502189081298787391521961515438835481\
6148499124506758522923769277844850343875938228889637921343518524059624407823654286240270359898184902896179851942792695748961300\
55465819813456262537001200841990445876541213338112
> convert(
  1085616295856442871891505512927230887846341430515401272513892546018586332106511445791302654568850218908129878739152196151543883\
  5481614849912450675852292376927784485034387593822888963792134351852405962440782365428624027035989818490289617985194279269574896\
  130055465819813456262537001200841990445876541213338112, hexadecimal)
F75AB314CBAADCCCD5894116664F5AFE97F260A1D3791B01221C464E7E69C226603DA0B3C78E3EB8CE84F029DC1ACB7BB18ECF6DB2\
15F6C1CB24CC4C78E57FEB4A96FDCFE6776D52C8F8CBDF962C8164A0C2642C2A5471D25279690212E820ECB3961ED2119E88A9A8\
04DFC488A144D981F968FA20A570DF19067A2B50B6A00
```

Figura 4.11. Resultados de la encriptación de un mensaje de 1024 bits usando Maple.

La Figura 4.12 muestra los resultados de la verificación en *hardware* para la encriptación usando SignalTap. El mensaje de 1024 bits es almacenado en cuatro


```

> convert(
  "2d2ff567b3fe74e06191b7fde6de112290c670692430d5969184047da234c9693deed1673ed429539c969d372c04d6b47e0f5b8cee0843e5c22835dbd3b05a0997a
  984ae6058b11bc4907cbf67ed84fa9ae252dfb0d0cd49e618e35dfdf59bca3ddd66c33cebbcc77ad441aa695e13e324b518f01c60f5a85c994ad179f2a6b5f9e93402a
  b11767be01bf073444d6ba1dd2bca5bd074d4a5fae3531ad1303d84b30d897318cbbba04e03c2e66de6d91f82f96ea1d4bb54a5aae102d594657f5c9789553512b29
  6dea29d8023196357e3e3a6e958f39e3c2344038ea04b31edc6f0f7ff6e7181a57c92826a268f86768e96f878562fc71d85d69e448612f7048f2d2ff567b3fe74e061
  91b7fde6de112290c670692430d5969184047da234c9693deed1673ed429539c969d372c04d6b47e0f5b8cee0843e5c22835dbd3b05a0997984ae6058b11bc4907a
  cbf67ed84fa9ae252dfb0d0cd49e618e35dfdf59bca3ddd66c33cebbcc77ad441aa695e13e324b518f01c60f5a85c994ad179f2a6b5f9e93402b11767be01bf073444
  d6ba1dd2bca5bd074d4a5fae3531ad1303d84b30d897318cbbba04e03c2e66de6d91f82f96ea1d4bb54a5aae102d594657f5c9789553512b296dea29d8023196357
  e3e3a6e958f39e3c2344038ea04b31edc6f0f7ff6e7181a57c92826a268f86768e96f878562fc71d85d69e448612f7048f2d2ff567b3fe74e06191b7fde6de112290
  c670692430d5969184047da234c9693deed1673ed429539c969d372c04d6b47e0f5b8cee0843e5c22835dbd3b05a0997984ae6058b11bc4907cbf67ed84fa9ae252
  dfb0d0cd49e618e35dfdf59bca3ddd66c33cebbcc77ad441aa695e13e324b518f01c60f5a85c994ad179f2a6b5f9e93402b11767be01bf073444d6ba1dd2bca5bd074
  d4a5fae3531ad1303d84b30d897318cbbba04e03c2e66de6d91f82f96ea1d4bb54a5aae102d594657f5c9789553512b296dea29d8023196357e3e3a6e958f39e3c23
  44038ea04b31edc6f0f7ff6e7181a57c92826a268f86768e96f878562fc71d85d69e448612f7048f2d2ff567b3fe74e06191b7fde6de112290c670692430d5969184
  047da234c9693deed1673ed429539c969d372c04d6b47e0f5b8cee0843e5c22835dbd3b05a0997984ae6058b11bc4907cbf67ed84fa9ae252dfb0d0cd49e618e35df
  dfe59bca3ddd66c33cebbcc77ad441aa695e13e324b518f01c60f5a85c994ad179f2a6b5f9e93402b11767be01bf073444d6ba1dd2bca5bd074d4a5fae3531ad1303d
  84b30d897318cbbba04e03c2e66de6d91f82f96ea1d4bb54a5aae102d594657f5c9789553512b296dea29d8023196357e3e3a6e958f39e3c2344038ea04b31edc6f
  0f7ff6e7181a57c92826a268f86768e96f878562fc71d85d69e448612f7048f", decimal, hexadecimal)
192531269695183965587312000360629602728449897010640074008113896363572745936669334825583863553336628518508534214168849113120461391002700
16711265262684936491236956063310315196728658915289986385965143275108998417906862488422086915688937501787639638993353939951672576897a
60609060932504558516527797442131629341933193129949393803671941715469496414780008900818024186124339004669510118847642882252121988637a
73418593066657510767454781969570863241566502639695079910191684863021209655688921596314643593062819888872168074497440115854324098321a
02522918859591837209186359105976867917833439713381358592062550443684374354979602770044926556241165384423745810620158581235721804931a
03872711168935441941016323789382392552643670893879195755133309431767176017514204638814803556829068532595169636234427599113609488265a
15766623022564218344450148397805341331183465131043517582485715895165787238191538895629051741588544838690614693284330580326353885425a
35127916030905384990484947513611908150578210203161931488570530239034402756068110091389485141381804580618212894116127014064268966207a
79446954963138861211012893324071210441561752665333515625740493656107893893774283003947515043458987439040056548899805423979725632374a
7076395574487374741213025065007867892250211614551073716844517961252644465474231443503543403618765923796382394955439799272592823775a
56491320617109584083866845213174080246569078848804502744026258331671547589970151366915895603653092265840655276249467993206735723157a
9529202236013004444776902326704698912191613385761003781198741711823035182377298532750366682020555128154311122426750023789004270376a
07801494830567495015724893160432176904396984132924959760656140877470444027760251618639830632876664745351127777186132545100363781492a
0423048323399272448038939605511036562639247529101329868238310520947055851285160736889753983248096312758073310558752960590416229175a
49774417120148420605529450046580021923001235140166130887053601315982334070349640455169800765856589097889526253669396979092257501889a
89967185518827680170753942393171314211500906836124452284293642703827684203714189197792575262157126630301785336374537812429365373683a
38863800284112192724634204525023847976096047795354322096307583985175075028041958977651857715120037868634492023265047104353485128739a
11287022095548898016326916827556543366843116589649145587583360013478392916783206551815500150972836666235590534873493472623748148044a
18256671807744202443183558346627019626266548733511434723602466250920178679894569187881014509261135742095

```

Figura 4.16. Resultados de la conversión hexadecimal-decimal del exponente d usando Maple.


```

> convert(
  "a5dd867ac4cb02f90b9457d48c14a770ef991c56c39c0ec65fd11afa8937cea57b9be7ac73b45c0017615b82d622e318753b6027c0fd157be12f8090fee2a7adcd0e1
  ef759f88ba4997c7a42d58c9aa12cb99ae001fe521c13bb5431445a8d5ae4f5e4c7e948ac227d3604071f20e577e905fbeb15dfa06d1de5ae6253d63a6a2120b31a5
  da5dabc9550600e20f27d3739e2627925fea3cc509f21dff04e6eea4549c540d6809ff9307eede91fff58733d8385a237d6d3705a33e391900992070df7adfl357cf7e1
  3700ce3667de83f17b8dfl778db381dce09cb4ad058a511001a738198ee27cf55a13b754539906582ec8b174bd58d5d1f3d767c613721ae05a5dd867ac4cb02f90b9457d48c14a770ef991c
  457d48c14a770ef991c56c39c0ec65fd11afa8937cea57b9be7ac73b45c0017615b82d622e318753b6027c0fd157be12f8090fee2a7adcd0eef759f88ba4997c7a42d
  58c9aa12cb99ae001fe521c13bb5431445a8d5ae4f5e4c7e948ac227d3604071f20e577e905fbeb15dfa06d1de5ae6253d63a6a2120b31a5da5dabc9550600e20f2
  7d3739e2627925fea3cc509f21dff04e6eea4549c540d6809ff9307eede91fff58733d8385a237d6d3705a33e391900992070df7adfl357cf7e3700ce3667de83f17b8
  dfl778db381dce09cb4ad058a511001a738198ee27cf55a13b754539906582ec8b174bd58d5d1f3d767c613721ae05a5dd867ac4cb02f90b9457d48c14a770ef991c
  56c39c0ec65fd11afa8937cea57b9be7ac73b45c0017615b82d622e318753b6027c0fd157be12f8090fee2a7adcd0eef759f88ba4997c7a42d58c9aa12cb99ae001fe
  521c13bb5431445a8d5ae4f5e4c7e948ac227d3604071f20e577e905fbeb15dfa06d1de5ae6253d63a6a2120b31a5da5dabc9550600e20f27d3739e2627925fea3cc
  c509f21dff04e6eea4549c540d6809ff9307eede91fff58733d8385a237d6d3705a33e391900992070df7adfl357cf7e3700ce3667de83f17b8dfl778db381dce09cb4
  ad058a511001a738198ee27cf55a13b754539906582ec8b174bd58d5d1f3d767c613721ae05a5dd867ac4cb02f90b9457d48c14a770ef991c56c39c0ec65fd11afa8
  937cea57b9be7ac73b45c0017615b82d622e318753b6027c0fd157be12f8090fee2a7adcd0eef759f88ba4997c7a42d58c9aa12cb99ae001fe521c13bb5431445a8d
  5ae4f5e4c7e948ac227d3604071f20e577e905fbeb15dfa06d1de5ae6253d63a6a2120b31a5da5dabc9550600e20f27d3739e2627925fea3cc509f21dff04e6eea45
  49c540d6809ff9307eede91fff58733d8385a237d6d3705a33e391900992070df7adfl357cf7e3700ce3667de83f17b8dfl778db381dce09cb4ad058a511001a73819
  8ee27cf55a13b754539906582ec8b174bd58d5d1f3d767c613721ae05", decimal, hexadecimal)
706708214848540790988023555273717115826417480208704725994553778139666218853229267951659058329845486133744325964873765340152722384913581 (
52877772635502684635593065586892275171413711911919114944523654810877841962214682399603366669103357531975498558946340539506161789584
60950238109919138116311420704303150028411740581539060869627968057282221243203534012353869197686330682082122573606607810462570357251
75709266457438165287391033928435020361515353845447324199761906117898627223746331453248136935553067422410609686834650480309162487068
54577213878294198615470312371223826608890153871409827324291302286628174721177579262809020256209249254104527435441419974661909470080
67228575052112920861365799747583991612942051536462990641660438554918528344038107779489412209672509127628640881731055161917676609287
55164051159076460540471237989413046718846021994281185648072716760637990093620904773947459612620518720747227771858372581826724617590
91124955557015656055047399569305528327052136740210401780770093093759474435295858286530411998296108191991732756203665449296689076662
31286511355719158330683419221459626275044658415905579328113721411708160833386801989221018968886394187986174974704437959253214286645
89141592866962818935503263135430405430203893277635078914790507612323712740606159678713446726769470910967785348115223405055155644483
19610789128478476178652307139589471501673880262984869484963895730192606097342410929234144510135508525451373693536432790415660319711
8154063143782804170818662089804616560412138963104605832983991462082422591206259367668271296849901358865874822326005215549753328400
91143354737200410688309314317021600938303704617161658852109822928000405298629125571717042452706530118614420445795117985020624930157
7552820763110301159315508442253352450825983133949828647604152473801648819965984012366068955537427286226641300282897488707423466077
75066100941533551088639507309899090254670048080536639759391889648371618213404442463053545675034376129803485438486685403254471372277
86798990947564773385591263918179813297449147380933325197592809696220534381571899448218012644474396574951721412722921774547156111360
65206360681026839333072054936011397171912932762964832627736793864592091616053294048916517764054852914027591489676397349664930340994
08125697890598834309110395810357228999956486755089386882892186498552784730773730310074540291074949022199108238100755767789188720800
1853975393345713318982982787716364484927378666357920402005557749637732607709540421665200459697465961989

```

Figura 4.17. Resultados de la conversión hexadecimal-decimal del módulo N usando Maple.

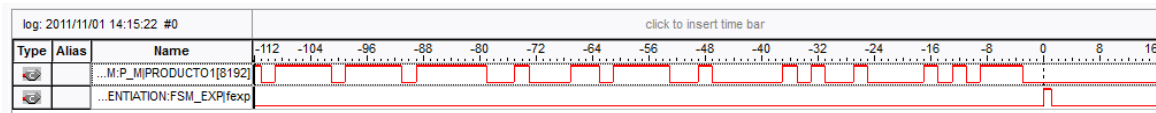

```

> convert(
498660016834795466108744756621642161909536787801446908435967724996391648968941839835775366275768989004685211362378981629168491667A
0594247387963902236619056690249686060326163389984736943954480257160881098418123360223066382339165276499355044414418798087286483593A
3811625198938584494396214695233564273827957255221696237977787894217518223189320951270170222328876097264673798622303894796910235934A
5946621067853799238652265719775581097267410390613006410527177079464976602907295445659365751912179214894108234422421206290940279751A
2595817137752878357101609898659593920074947608305846074892882349342776495802976235639134227022663120984116030523719098177408357473A
0511291500711321495414404896267617815273618209743492258457545641985483882935611171185928501866024867353447140797467235143874023939A
43305960318308854673006892395353375114178893996366556392362627311058929163006091551679489964984894624336105980736596837498408536A
0335353394599447536590761959061604507667221003154273335141524330127061847491859747426252298847194358662154023930171355176624609855A
1063261704059795340359214520455425949888582146973988597940826503957153131491261277120912536608867013767081542331035478084033364724A
8950010264639601798092170760489107060149765189974828938560873905006611781726914515391738314399005673210722234830365532550739108995A
1322214276325952399169413715859134789212283002893461406685245204935326987708267801566684997989914688186322513442705265482688207086A
4143310071593391922611416974402258614351783582643628459255619010500110999563502866979830618141285561595102210269260120551848044640A
8764937203955908130370638461223667014424431331950946182021220384381049073630348011528678889691651003243315897184069804794619840363A
613858280778688330446477340558359847607689343675217016597670497073416060802151710447334280911625721630528227345574471278816114804A
7906642448546667487820377583671634861770497319399323473634709447617847050703700180889312845785976024116656086837867172861078116192A
2171358002371634818501170311549532205640078990574315682676106339643991738767576930441485777183085292860476257430632829989382368595A
764379425819236488558676487864409143852275469107961793954326916674787286428218580702150308910249184713365910218933416370792981319A
1530053909969881054691255471380688774615931823091280615555052396030248218448609272015602428979983374709148775169272271153496794924A
3074695663362219171567690775247204279702735709517561797392109195661264459537199958861017110310948181252975360535803331685852,
hexadecimal)
7509413D89P7BD179D675266B90EFA857A232D55D827BE4PCA005766A6DD8B325501766735297FFB45C9CE9236DF020D51862093A38B969A
DF1FD57798D1AEB1ABF157A5804215F773A22FF8F3A4006F0DC2BBDA4193A3926654A7DB9EBBCDA1CFCB923A28B0388FA9F87BED3F
17242613D4B467BB2788BB1DDBEE77B10C077F50F1551919B37CB84D90A12E8D6F3607340DD5313E52875C7377DEA3020EB8D29A189CA
36053B71C353737ACECBFF7006F63E929AB4B3A13C209B801B2FE1E6B878770FDF8F5DA5EE319B7FADDF6139AD6C6394BE2D6A75D2E8A
7E86B0FDE282960293005FF04D1A844D0BA42AC955E4494DE2C94C6EA78F2056BDB1B94090D48236ADS251A629BE1D70077EDD9793A
F4A80D2B99E6315BFBF6C730B50B97097469082961A7FBE41A2DCECB10A81A6FAD98A217C8056494E808C3862AA2BC91449F9A03F4C9A
5B40CA3C059A67BB67340EB4DC9368F6C76688D396FCA8F821E3FE49A6E52012167420EC34EAEAOAF6514F6BE6B773E99469D0D86BD1A
B72B9384583C0AE93998E60D565C231E8DD57706A7608D17264BAE23CA4CDC5E59DA25F17829FB0F12755E903C91FF94865E8B7B131B
07AA01F03C4BFACD400017EEFBED469BD8CC8513245C8A8F4A36D2126BD8CB14FF26A4500FEC8F647A7FC2DD4EFC04BFFB17284A2A
D6116A1DBE47C21300CD7092D2499B8B142BB6DAA614B0F54A6AD0EB1F27855B0E3EC91628E38AC3F5B73EDFC1EE71B439ED356D70A
252B5F0B604116CB454AB31DD5D14F27D68E23148721608E09DFD448AF9822CFFCD4528AD764DB9A13BBE36FDP38B2F6E26678D288EF
5A9BC22A7A34SD1904C69A4E44F8FDBFD9B51BB9CB6A2158B35D4D351F2AB0708E6D2F0CD802B3C680A2E018336C18195D9D5B07B
0BB60768AEE5345500A418AB125F85A02AAE5C04256166F226A55F89F8E86B641F5B3F7916AC8E5C705D47AFBFB0B8E453E2A040AA3EB
B4638E444F2F1B50353D4D2349522AC921FDBE1A154B848373DF2D199340AEAA46056C53588B2D74695FD477A778B0B43AEB721514579A
6E84639B15B72A6A515AB15C8BA7E57DC7D4BCB756C5B9931A7EB25281F7EB127077F28154BDD873043B414BB0E9BFC1AEC2BFF9F7762A
CF46F0379B9B8779E29F65C9D4173A2FF9B3014D576B1E7378FC554BD57EDB052AF7C3C2DFD526FB6504EC553F7067EEFB2B21C8CA
B990E296595C636DEB3F98D6AF51CB51750FEBB28BC5D149BAA9F7B491F57D73CD729DEBA9869B15D6849915D4205BFC646PDCBA7
F5B5FE23C9FBE3C36A76EDCF0ED8CB86F9CDB2A1336BD40B178C73237FA58A71223B4E891AD47BD2A76F5B08FB0F96DAF8430AFB9A
B626898FCB271A5615C96B311D7B2B7D607130165D8A2211D118BBF25B62C61FE72A83BFD138EE5B56AA17E04DF2A46230197045216A
4F6EDDC

```

Figura 4.19. Conversión decimal-hexadecimal del resultado de la exponenciación modular de 8192 bits.

La Figura 4.20 muestra los resultados de la verificación en *hardware* para la exponenciación modular de 8192 bits usando SignalTap, el cual se configuró con dos señales: el disparo, *fexp*, y el bit más significativo de *PROD1* (ver Figura 3.20), debido a que el resultado se puede verificar en esta señal, desde el bit menos significativo hasta el bit más significativo cada dos ciclos de reloj, 16 K ciclos antes de que *fexp* sea '1'. En este caso, se presentan los 56 bits más significativos del resultado. Estos resultados son comparados con los obtenidos en Maple (ver Figura 4.19), y desde esta Figura se puede observar que los resultados son iguales.



01110101000010010100000100111101100010011111011110111101

Figura 4.20. Resultado de la verificación *hardware* del Criptoprocador RSA basado en el multiplicador de Montgomery base 2 con la exponenciación modular de 8192 bits.

4.2.2. Verificación en *hardware* del Criptoprocador RSA basado en el multiplicador de Montgomery base 4

En esta sub-sección se presenta la verificación *hardware* del Criptoprocador RSA basado en el multiplicador de Montgomery base 4. En este caso se llevan a cabo dos pruebas: la encriptación y desencriptación de un mensaje de 2048 bits, y la exponenciación modular de un número de 8192 bits, usando un exponente y un módulo de 8192 bits.

La verificación en *hardware* del Criptoprocador RSA basado en el multiplicador de Montgomery base 4 que lleva a cabo la encriptación y desencriptación de un mensaje de 2048 bits se realiza usando los vectores de *test* estándar recomendados por RSA Labs [12]. Para los dos procesos se presentan los resultados de simulación obtenidos en Maple y la verificación en *hardware* usando SignalTap II ELA.

Para llevar a cabo el proceso de encriptación usando el procedimiento *exponenciación*, primero se efectúan las conversiones hexadecimal-decimal del módulo N y del exponente e (vectores de *test* de RSA Labs), las cuales se muestran en las Figuras 4.21 y 4.22, respectivamente. En la Figura 4.23 se muestran los resultados del procedimiento *exponenciación* para la encriptación y la conversión del resultado a hexadecimal, con el fin de realizar la comparación con los resultados obtenidos de la verificación en hardware.

```
> convert(
    "a5dd867ac4cb02f90b9457d48c14a770e991c56c39c0ec65fd11afa8937cea57b9be7ac73b45c0017615b82d622e318753b6027c0fd157be12f80\
    90fee2a7adcd0eeef759f88ba4997c7a42d58c9aa12cb99ae001fe521c13bb5431445a8d5ae4f5e4c7e948ac227d3604071f20e577e905fbeb15dfa\
    6d1de5ae6253d63a6a2120b31a5da5dabc9550600e20f27d3739e2627925fea3cc509f21dff04e6eea4549c540d6809ff9307eede91ffef58733d3835\
    a237d6d3705a33e391900992070df7adfl357cf7e3700ce3667de83f17b8df1778db381dce09cb4ad058a511001a738198ee27c55a13b754539906\
    582ec8b174bd58d5d1f3d767c613721ae05", decimal, hexadecimal)
20938558521519708868374048120183292514743246419346126258296019099489758353430955129638513694940428380396792060825129271360\
9736741886713186380350904524266239246959812062913729388014861738032125908945131791034341736470159837960962209406608188\
5793120505947421156979430883885490457761729448944249382710097827649868886164777782729964351080098562770731883228810419\
2155082517055602434187160611690322873947574057106057503742140492111084521038129922798577326080702449745442919636743356\
6332662529853668362299092702539491774418048113009063205071759007342269608192975827659156427717126340975269773387924025\
76964243519023177903621
```

Figura 4.21. Resultados de la conversión hexadecimal-decimal del módulo N usando Maple.

```
> convert("010001", decimal, hexadecimal)
65537
```

Figura 4.22. Resultados de la conversión hexadecimal-decimal del exponente e usando Maple.

```
> exponenciacion(2^2046 + 1, 2050, 65537, 17,
    209385585215197088683740481201832925147432464193461262582960190994897583534309551296385136949404283803967920608251292\
    713609736741886713186380350904524266239246959812062913729388014861738032125908945131791034341736470159837960962209406\
    608188579312050594742115697943088388549045776172944894424938271009782764986888616477778272996435108009856277073188322\
    881041921550825170556024341871606116903228739475740571060575037421404921110845210381299227985773260807024497454429196\
    367433566332662529853668362299092702539491774418048113009063205071759007342269608192975827659156427717126340975269773\
    38792402576964243519023177903621)
86156889842951709129804807654622208325156221757379321324202834658847524401899743458998440980343582616768361875142736490124\
5776179189131953738578724522682169040095997863839213970056761410428018419441002206523263893429458540111748436918919297\
958441761979951786431508473317389525541001567881888590888731512078056661430210525014268636044933501860012914649782216\
3557280915419906780916499021333756071270288257338814817217381815453061479857293086015346895761154067756378248005349502\
3013333107043521268928265897718750919745714110375077828989921595878528423650436176797574307751804921230680225499722195\
6681607203135772584592
> convert(
    861568898429517091298048076546222083251562217573793213242028346588475244018997434589984409803435826167683618751427364\
    901245776179189131953738578724522682169040095997863839213970056761410428018419441002206523263893429458540111748436918\
    91929795844176197995178643150847331738952554100156788188859088873151207805666143021052501426863604493350186001291464\
    978221635572809154199067809164990213337560712702882573388148172173818154530614798572930860153468957611540677563782480\
    053495023013333107043521268928265897718750919745714110375077828989921595878528423650436176797574307751804921230680225\
    4997221956681607203135772584592, hexadecimal)
443FD8E49AB89EBB52A7F7C166BF630CCF5273DD6EA157188FCE130773F2CF0619334983323421F7AA543E2D63165B52BBF3A\
C09B42673BBAF3EF9BE9888EFDD8BD63DA4839FD8FDC951E043E21CA08BA49056CD12FFCDA7811BA4B0FF3AD3DDBB93\
F456B6B44A1595768241834742C59500BF35FDA5A1FC67E945B58819FC9972377085FAC3C41BE415C3AB913208FF3D16A5\
5E7D66FEBB2A474539525A665DA7F41DBB68625BFB46BF55232A9C29AC8BBFAD46C52B25BE795AD89DDAC36F06D892\
02CAEA87226B5A437B3E20C29C42CAEB662F023A4BB7A889D027D1C1162B0D46F78D5AF9BAFF65660BC400F951F2158B3\
BF7A35EE3B465EEC1399BE6A90
```

Figura 4.23. Resultados de la encriptación de un mensaje de 2048 bits usando Maple.

La Figura 4.24 muestra los resultados de la verificación en *hardware* para la encriptación usando SignalTap, el cual es configurado con la señal de disparo

fexp. Después de que *fexp* es '1' se pueden visualizar los 48 bits más y menos significativos del resultado en *PROD1*[2047..2000] y *PROD1*[47..0], respectivamente. Estos resultados son comparados con los obtenidos en Maple (ver Figura 4.18), y desde esta Figura se puede observar que los resultados son iguales.

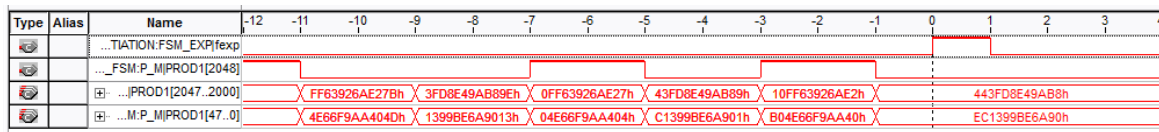


Figura 4.24. Resultado de la verificación en hardware del criptoprocador RSA basado en el multiplicador de Montgomery base 4 para la encriptación de un mensaje de 2048 bits

Para llevar a cabo el proceso de la descryptación usando el procedimiento *exponenciación*, se efectúa la conversión hexadecimal-decimal del exponente *d* (vector de *test* de RSA Labs), la cual se muestra en la Figura 4.25. En la Figura 4.26 se muestra el procedimiento *exponenciación* para la descryptación y la conversión del resultado a hexadecimal, con el fin de realizar la comparación con los resultados obtenidos de la verificación en *hardware*.

```
> convert(
  "2d2ff567b3fe74e06191b7fde6de112290c670692430d5969184047da234c9693deed1673ed429539c969d372c04d6b47e0f5b8cee0843e5c2283i
  5dbd3b05a0997984ae6058b11bc4907cbf67ed84fa9ae252d8b0d0cd49e618e35dfdf59bca3ddd66c33cebbc77ad441aa695e13e324b518f01c60f
  5a85c994ad179f2a6b5f9e93402b11767be01bf073444d6ba1dd2bca5bd074d4a5fae3531ad1303d84b30d897318cbbba04e03c2e66de6d91f82f90i
  ea1d4bb54a5aae102d594657f5c9789553512b296dea29d8023196357e3e3a6e958f39e3c2344038ea604b31edc6f0f7ff6e7181a57c92826a268f8i
  6768e96f878562fc71d85d69e448612f7048f", decimal, hexadecimal)
5704372997275945526807520059841198043921499201889244415807044463613529617183163531472768707238870844222641233196861402797i
5806743979175211899839056537301305924110649629566989846699017181392891350116858501765819999970469937115522493835586905i
5631669144672205173762797918173472905092817560488746906876640930127114041551417357574366143060890411800024647351946740i
0570596428951819671249184677305228164432693660562899787695302877558188644601445059894117717891865903310330267549552501i
7553048252374893091500467336277895984248078017287869750851517253945258343449826826509589521306620407209418477927837379i
4526872568508146189455
```

Figura 4.25. Resultado de la conversión hexadecimal- decimal del exponente *d* usando Maple.

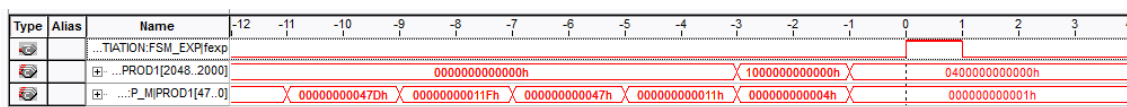


Figura 4.27. Resultado de la verificación en hardware del criptoprocesador RSA basado en el multiplicador de Montgomery base 4 para la descryptación de un mensaje de 2048 bits

Debido a que no hay vectores de test estándar para 8192 bits, la segunda prueba del Criptoprocesador RSA basado en el multiplicador de Montgomery base 4 está orientada a verificar la exponenciación modular de un número de 8192 bits usando un exponente y un módulo de 8192 bits. Se presentan el resultado obtenido en Maple y la verificación en *hardware* usando SignalTap II ELA. Los vectores de *test* usados para realizar la exponenciación modular de 8192 bits son presentados en el Anexo 1.

La prueba realizada en Maple consiste en llevar a cabo el procedimiento *exponenciación* usando los vectores de *test* del Anexo 1 (ver Figura 4.28). En la Figura 4.29 se muestra la conversión del resultado a hexadecimal, con el fin de realizar la comparación con los resultados obtenidos de la verificación en *hardware*.


```

convert(
1278134362614292255840278229019288617295075491497001162943157985285588979254570602117839624714083214262265768776243018048248484175822725137694539941882025097851974533828306142795673797350880316249
341387867807218841634141969333712795517517858634638954181746268646959713452921649445378828214262162058349459313955769843030129335781780944141709872599155287989199060357436190669037384316343147714
8691508846329368853869586008364135418599225617155222239748445114628196167244128121581072442048008738157103805915771687206987815792882968502751965136690348391713470869378165737385459567646850745
11992893858733460135511402775421253846525159436218628993681687449984498115778217214732767818874519040965335309105356575329787202400563111723373659812672330573384521103221272367673009211696180078
1477459593840112049661532899194929325908944080603108836432085291803166348056531286380895817613822155840609295902136255410633791054885447790953087596488034220238664935605466456696868989384851
63241414983403892829157700715705362242166324332777578511030431930124996610147126053420469237156083473071688935488072418201878485496905494398895363688255066052697958895
67919711948626343682712869451092480033610547248167840252922811169419111826936124140185058776552524210729197504454229719652705050709591839952586407150906935496866429760237577246077843231409
07716438151560783794265660054452806574379765356062062494500039587836584845354173587283809366556666271082800165371022872952320277349913166736719063980556676336670106381014457779641796381026830
9303279661772010237525262477011209692879749515131741605858049125070239195305851513868349659590948830624174763274025886375169289819101586281785765989866100853714939502345073119953133792105054710
4043787363293305432470226619218346643022152812385810055413106239235139000729174183503143085241704802008295110253468144740452194715585450056782342365327573045658984195069998520633797017878047159
2099768746603067394282691231990256467624736959115429666596828472931845806362408175934302139700993497767560487359079920030691137290826483442507269805001403044217825345660092902523362464975031855
401912615987910118380063805950199839491103950214730498434133017441063793949240736802032349462069316242807246927315930456929090261246860237238548130557444169331585875706976872279501600754893541870
768115830744362217460960892714590399535024376153442237887155571328300922587660083043239961484043738802846685055401102267947_hexadecimal)
IDFF7B8D7F60EBB3CC5B503F0BBF8EB669F21193EBE54674643CEAE72BF69B6436B54B6FDC082BF5E59F933340411E17D2ED57P669122A0606674142AB48EA09144189433DEB0FCE2ADA95ECC6682230907A
EB6BE2B466B662FF9870D31125D581256BA851389F6CA8490975016DA4AB288FFBEE9B4DEF51F54CC265C0B74C181424C077A59AC3FB47D19C21C3BC28EBDC1801DE7B9B493D7E2D3B897E662FD52A
B1463B2BD5052BF466BF2EDDA43D9D7ETB39150A7D6343D0P81B1019DA75B3DE0926E4B01EBDA4BDD4DBA9CA6A54445DD07D86606C274F4F8822C1233A7372727F123AB2225E8FFD81C7F8C8C6FB4FA
8DB27AC748B54A9298C3D3E749395D3DC0508E8E8A98D55A182201F78CD8134D9D587A047AD1B5933CCB8897F5C371B4B2AF643BD9D8300FF63F1E7D5434B58F6301D3A51496D8497A4P637C9PCD2A
ACB5CDCCE341DD125088C86235140A4E19E7003234069B4AC192D323E761AD1674E705DCD9CBEP936148B896A04C9CC1DD4D043D603040A4F730EFPF250354041363B0439592585CDAFE5FD414D63CB92A
B14509026F55CD5B3CB0CPCF45169D20DF90385544B810254CC68AABAD103D6687B17E686912BB405EP4C5A9B483D6FC65P6AADI F6434945D82C34A4C42831428C1C86D6B4115EB714C7D50416F87CC
E9A849AB6C433B9B98D0864C96B6BF93AB26F18AEBF193CF48582EBE1743E56BC0B92D9D3830CB83303F3EP451187D82EDC57979EBEEB1615E344606BB8C10FF4BFD7F38E2A334A2DF8E1AF78649048F9A
12P724ABE68461E7497EAB1E70301FD03223EB9DC2E081BD7AB5BA71BA101B6B5CE70F15375373B927F80B70F24993E2BDE3FAB14C37AD54FE1881623FC0B6323847421504C2200FCE9C9F6AD7F2D0F4A
F58F277EB02D44EEDD924BCB8940A58DD252F967792F8760EE9910B2F677CFF52076403CCF236C3DF4411071D9B77983221960574BGE0870EBA1D146327676530B1D608413B51B776C453956008BD9A20C
A7B9B48C7C87DC25C91F9D6140B243D96C7375BAE2BCF7CAB49399845615A13A1128CF421628C2C4C43B44365419B35EA3F139CA465C3622489FB8CCCD9BA36427CB612633D405E7BF4E22F2D60B93A
74DF3C675EC6954F70008220DC9275DC4B649BA813F5308EBBF12E4CC210D40A2E8860EP9148BEDD5DBBC390252934C84E2D5E9482980A6DFE2FB69E7639378501DA71705B4593FFD8436AD816920A4620A
9C10EB4E19802DA681CA8862C7FE132C27D33BE23204BD4AC7170691D1F59AB99CBE6E98A1BC06367172775529ACA18BF6E1601E4FDF2DF4432F1F336716DAB8BC078D1DD0526CCBFAD7978EB68A
0827EAD3B8C887E546F5B15993EBDA1172C54496E3CC26799AB1F9128AD17357AA404FEDD25F516C1FAE9DD169179B25760A6B8BD4630D8792C4BF8B1E1D62B

```

Figura 4.29. Resultados de la conversión de la exponenciación modular de 8192 bits a hexadecimal usando Maple

En el Anexo 2 se muestran los resultados de la verificación en *hardware* de la exponenciación modular de 8192 bits. El SignalTap se configuró con tres señales: el disparo, *fexp*, y los dos bits más significativos de *PROD1* (ver Figura 3.26), debido a que el resultado se puede observar en estas dos señales como dígitos base 4, cada dos ciclos de reloj, desde el menos significativo hasta el más significativo, 8 K ciclos antes de que *fexp* sea '1'. El orden en que se presentan las *waveform* es desde las que muestran los últimos dígitos base 4 hasta las que muestran los primeros dígitos base 4 que se obtuvieron en *PROD1*[8193..8192], es decir, desde los dígitos base 4 más significativos hasta los menos significativos del resultado. Debajo de cada *waveform* se presentan los dígitos base 4 del resultado desde los más significativos hasta los menos significativos. Debido a que la comparación entre los resultados de Maple con los obtenidos desde el SignalTap se llevó a cabo en forma manual, se verificaron aproximadamente 4 K bits del resultado, el cual es un número de bits suficiente para inferir que el resultado obtenido es correcto.

4.3.RESULTADOS DE SÍNTESIS

En esta sección se presentan los resultados de síntesis para los multiplicadores de Montgomery base 2 y 4, y los Criptoprocesadores RSA basados en los multiplicadores anteriores, para tamaños de clave de 512, 1024, 2048, 4096 y 8192 bits. En este caso se realiza un análisis de las relaciones entre recursos requeridos vs tamaño de clave y entre frecuencia máxima de operación vs tamaño de clave, con el propósito de determinar cuál es la tendencia de los recursos requeridos y la frecuencia máxima de operación a medida que aumenta el tamaño de la clave.

Las Tablas 4.1 y 4.2 presentan los resultados de síntesis en el FPGA Stratix III de los multiplicadores de Montgomery base 2 y 4, respectivamente, para diferentes tamaños de clave. Usando los datos presentados en las Tablas 4.1 y 4.2, la Figura 4.30 muestra que existe una relación directa entre el tamaño de la clave vs área para los multiplicadores de Montgomery base 2 y 4; se observa también que el

multiplicador de Montgomery base 4 requiere más recursos que el multiplicador de Montgomery base 2, por ejemplo: para una clave de 8192 bits los recursos del multiplicador base 4 son aproximadamente 2,57 veces los del multiplicador base 2. Con los datos de las Tablas 4.1 y 4.2 también se obtiene la Figura 4.31, en la cual se puede observar una relación inversa entre el tamaño de la clave y la Frecuencia Máxima de Operación (*FMO*) para los multiplicadores de Montgomery base 2 y 4. En este caso, la *FMO* es mayor para el multiplicador de Montgomery base 2, por ejemplo: para una clave de 8192 bits, la *FMO* del multiplicador base 2 es mayor que la *FMO* del multiplicador base 4 en un factor de 1.41.

TABLA 4.1: RESULTADOS DE SÍNTESIS DEL MULTIPLICADOR DE MONTGOMERY BASE 2 PARA DIFERENTES TAMAÑOS DE CLAVE EN EL FPGA EP3SL150F1152C2

N (Bits)	AREA				FRECUENCIA MAXIMA (Mhz)
	ALUTs	REGs	BLOCK MEMORY BITS	% LÓGICA USADA	
512	1620	2627	506	2	432.71
1024	3164	5191	1018	4	474.38
2048	6242	10315	2042	9	398.09
4096	12392	20559	4090	17	402.9
8192	24683	41043	8186	34	369.96

TABLA 4.2: RESULTADOS DE SÍNTESIS DEL MULTIPLICADOR DE MONTGOMERY BASE 4 PARA DIFERENTES TAMAÑOS DE CLAVE EN EL FPGA EP3SL150F1152C2

N (Bits)	AREA				FRECUENCIA MAXIMA (Mhz)
	ALUTs	REGs	BLOCK MEMORY BITS	% LÓGICA USADA	
512	2357	3456	757	3	492.85
1024	4550	6794	1525	6	363.5
2048	8922	13462	3061	11	419.99
4096	17641	26782	6133	23	342.23
8192	35067	53417	12277	45	301.57

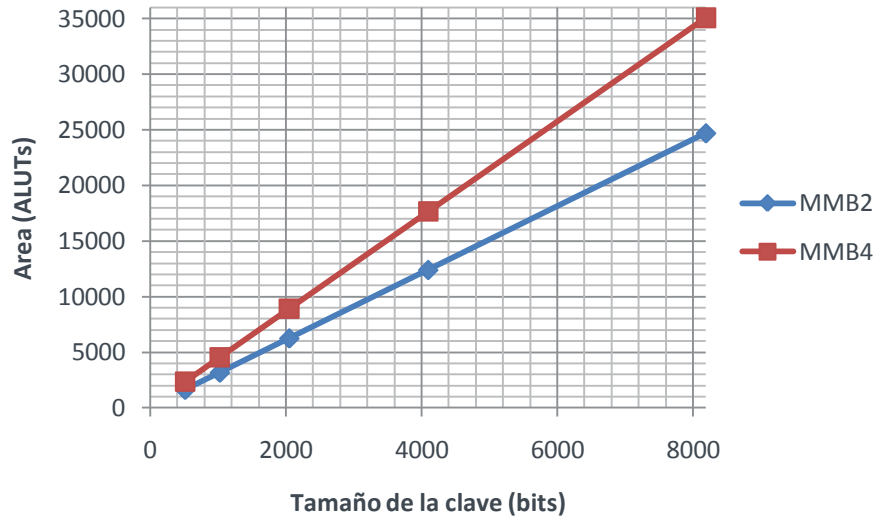


Figura 4.30. Área vs tamaño de clave para los multiplicadores de Montgomery base 2 y base 4.

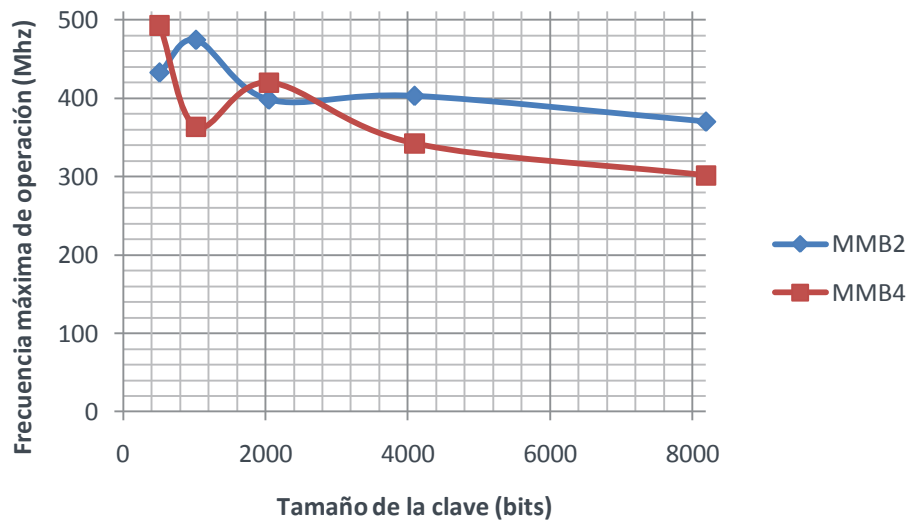


Figura 4.31. Frecuencia máxima de operación vs tamaño de clave para los multiplicadores de Montgomery base 2 y base 4.

Las Tablas 4.3 y 4.4 presentan los resultados de síntesis en el FPGA Stratix III de los Criptoprocesadores RSA basados en los multiplicadores de Montgomery base 2 (*MMB2*) y 4 (*MMB4*), respectivamente. Usando los datos presentados en las Tablas 4.3 y 4.4, la Figura 4.32 muestra que hay una relación directa entre el tamaño de la clave vs área. El Criptoprocesador RSA basado en el *MMB4*

consume más recursos que el Criptoprocador RSA basado en el *MMB2*, por ejemplo: para una clave de 8192 bits, el Criptoprocador RSA basado en el *MMB4* consume 2.88 veces más recursos que el Criptoprocador RSA basado en el *MMB2*. Con los datos de las Tablas 4.3 y 4.4 también se obtiene la Figura 4.33, la cual muestra una relación inversa entre el tamaño de la clave y la *FMO*. En este caso, para los tamaños de clave 512, 1024, 2048 y 4096, la *FMO* es ligeramente mayor en el Criptoprocador RSA basado en el *MMB2*, y para el tamaño de clave de 8192 bits la *FMO* es mayor en el Criptoprocador RSA basado en el *MMB4*.

TABLA 4.3: RESULTADOS DE SÍNTESIS DEL CRIPTOPROCESADOR RSA BASADO EN EL MULTIPLICADOR DE MONTGOMERY BASE 2 PARA DIFERENTES TAMAÑOS DE CLAVE EN EL FPGA EP3SL150F1152C2

N (Bits)	AREA				FRECUENCIA MAXIMA (Mhz)
	ALUTs	REGs	BLOCK MEMORY BITS	% LÓGICA USADA	
512	2118	5205	509	4	373.83
1024	4127	10328	1021	8	380.66
2048	8150	20572	2045	17	311.43
4096	16193	41059	4093	33	290.53
8192	32262	82023	8189	66	235.46

TABLA 4.4: RESULTADOS DE SÍNTESIS DEL CRIPTOPROCESADOR RSA BASADO EN EL MULTIPLICADOR DE MONTGOMERY BASE 4 PARA DIFERENTES TAMAÑOS DE CLAVE EN EL FPGA EP3SL150F1152C2

N (Bits)	AREA				FRECUENCIA MAXIMA (Mhz)
	ALUTs	REGs	BLOCK MEMORY BITS	% LÓGICA USADA	
512	4988	6006	251	6	332.56
1024	9827	11907	507	12	321.23
2048	19490	23710	1019	24	317.36
4096	38796	47253	2043	49	270.86
8192	79450	89062	4091	93	263.44

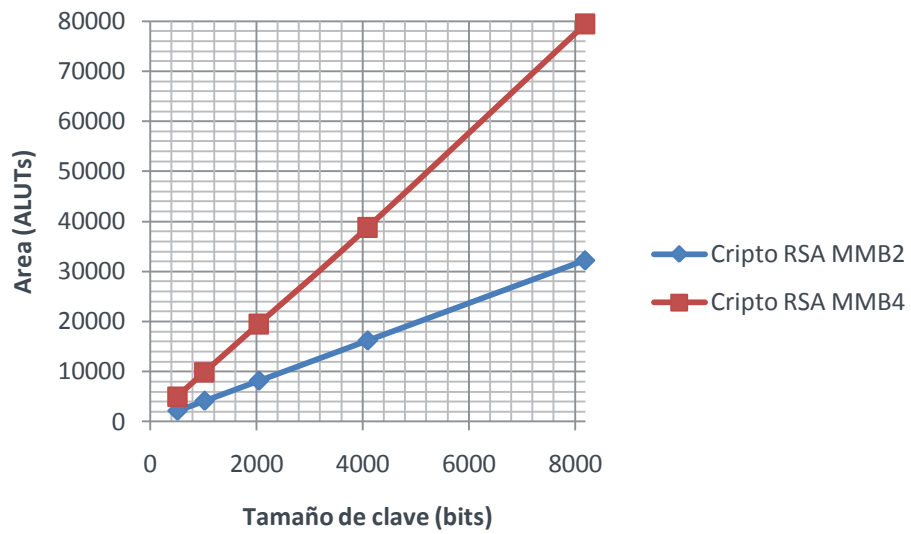


Figura 4.32. Área vs tamaño de clave para los Criptoprocesadores RSA.

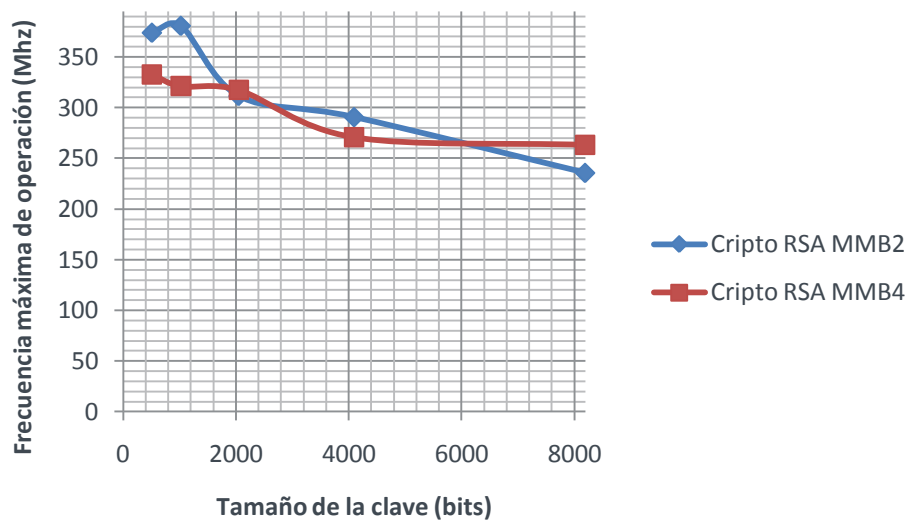


Figura 4.33. Frecuencia máxima de operación vs tamaño de la clave para los criptoprocesadores RSA

La Tabla 4.5 presenta el tiempo de ejecución calculado para la multiplicación de Montgomery base 2 y 4, usando la *FMO* de los Criptoprocesadores basados en el *MMB2* y el *MMB4*, y las Ecuaciones 4.1 y 4.2, respectivamente, donde n y nb son el número de bits del módulo de los Criptoprocesadores basados en el *MMB2* y el *MMB4*, respectivamente. Adicionalmente, se puede verificar desde la Tabla 4.5 que T_{MMB2} es mayor que T_{MMB4} en un factor promedio de 1.92.

$$T_{MMB2} = 4*(n+2)*(1/FMO) \quad (4.1)$$

$$T_{MMB4} = 4*(nb/2+1)*(1/FMO) \quad (4.2)$$

TABLA 4.5: TIEMPO DE EJECUCIÓN PARA LA MULTIPLICACIÓN DE MONTGOMERY BASE 2 Y 4 USANDO LA *FMO* PARA DIFERENTES TAMAÑOS DE CLAVE

N	$T_{MMB2}(us)$	$T_{MMB4} (us)$	T_{MMB2}/T_{MMB4}
512	5.5	3.09	1.78
1024	10.78	6.39	1.68
2048	26.33	12.92	2.04
4096	56.42	30.26	1.86
8192	139.2	62.21	2.24

Usando los datos de la Tabla 4.5, la Figura 4.34 muestra una relación de proporcionalidad directa entre el tiempo de ejecución vs el tamaño de la clave para la multiplicación de Montgomery base 2 y 4. La multiplicación de Montgomery base 4 se lleva a cabo más rápido que la multiplicación de Montgomery base 2, por ejemplo: para una clave de 8192 bits el tiempo de ejecución para la multiplicación de Montgomery base 2 es mayor que el tiempo de ejecución para la multiplicación de Montgomery base 4 en un factor de 2.33.

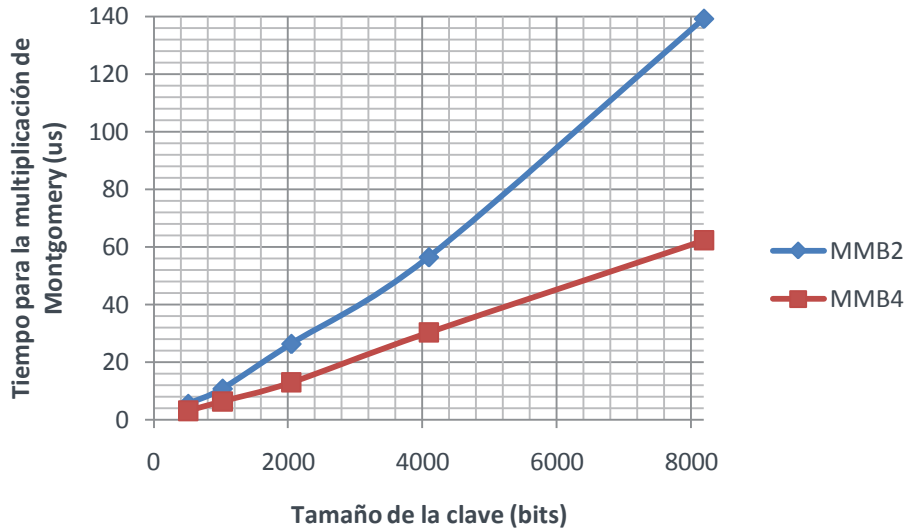


Figura 4.34. Tiempo de ejecución vs tamaño de clave para la multiplicación de Montgomery

La Tabla 4.6 muestra el tiempo de ejecución calculado para la exponenciación modular (desencriptación) usando la *FMO* de los Criptoprocesadores RSA basados en el *MMB2* y el *MMB4*, y las Ecuaciones 4.3 y 4.4, respectivamente, donde n y nb son los bits del módulo para los Criptoprocesadores RSA basados en el *MMB2* y el *MMB4*, respectivamente. Adicionalmente, se puede verificar desde la Tabla 4.6 que T_{EXP1} es mayor que T_{EXP2} en un factor promedio de 1.92.

$$T_{EXP1} = (n+2) * T_{MMB2} \quad (4.3)$$

$$T_{EXP2} = (nb+2) * T_{MMB4} \quad (4.4)$$

TABLA 4.6: TIEMPO DE EJECUCIÓN PARA LA EXPONENCIACIÓN MODULAR (DESENCRIPTACIÓN) DE LOS CRIPTOPROCESADORES RSA BASADOS EN EL *MMB2* Y EL *MMB4* USANDO LA *FMO* PARA DIFERENTES TAMAÑOS DE CLAVE

N	$T_{EXP1}(ms)$	$T_{EXP2}(ms)$	T_{EXP1}/T_{EXP2}
512	2.83	1.59	1.78
1024	11.06	6.56	1.68
2048	53.98	26.49	2.04
4096	231.21	124	1.86
8192	1140.60	509.75	2.24

Usando los datos de la Tabla 4.6, la Figura 4.35 muestra una relación de tipo exponencial entre el tiempo de ejecución y el tamaño de la clave para la exponenciación modular. El Criptoprocador RSA basado en el *MMB2* realiza la exponenciación modular en más tiempo que el Criptoprocador RSA basado en *MMB4*, por ejemplo: para una clave de 8192 bits el tiempo de ejecución para la exponenciación modular del Criptoprocador RSA basado en el *MMB2* es mayor que el tiempo de ejecución para la exponenciación modular del Criptoprocador RSA basado en el *MMB4* en un factor de 2.23.

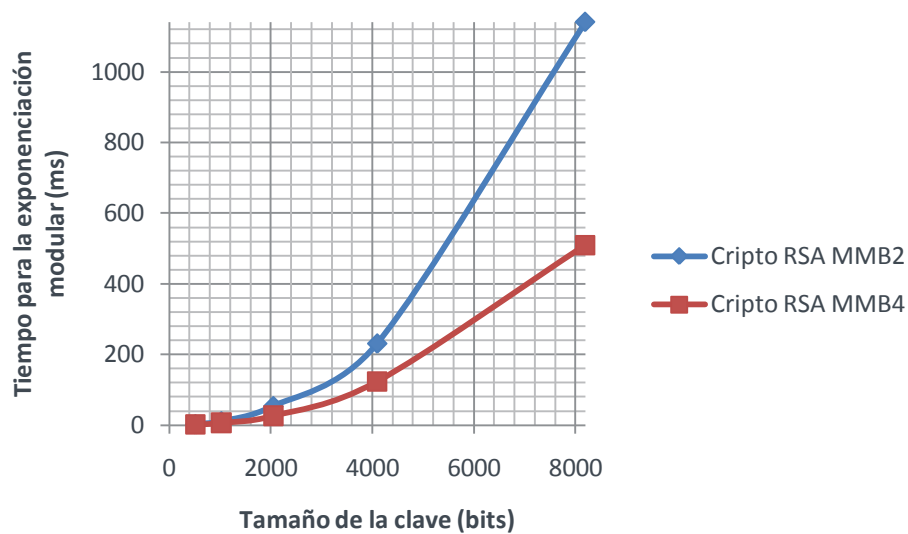


Figura 4.35. Tiempo de ejecución vs tamaño de clave para la exponenciación modular.

Con el propósito de evaluar la eficiencia de los Criptoprocadores RSA basados en el *MMB2* y en el *MMB4*, los tiempos de ejecución para la exponenciación modular en *hardware* son comparados con los obtenidos para el *software*. La Tabla 4.7 presenta el tiempo para la descryptación usando Maple, y los Criptoprocadores RSA. La Figura 4.36 muestra que las dos implementaciones *hardware* son más veloces que su contraparte en *software*, corriendo en un *PC* con un procesador I7 y 16 GB de RAM.

TABLA 4.7: TIEMPO DE EJECUCIÓN PARA LA DESENCRIPTACIÓN USANDO MAPLE, Y LOS CRIPTOPROCESADORES RSA BASADOS EN EL *MMB2* Y EL *MMB4*

N	$T_{MAP}(s)$	$T_{EXP1}(ms)$	$T_{EXP2}(ms)$
512	0.05	2.83	1.59
1024	0.11	11.06	6.56
2048	0.31	53.98	26.49
4096	1.23	231.21	124
8192	5.93	1140.60	509.75

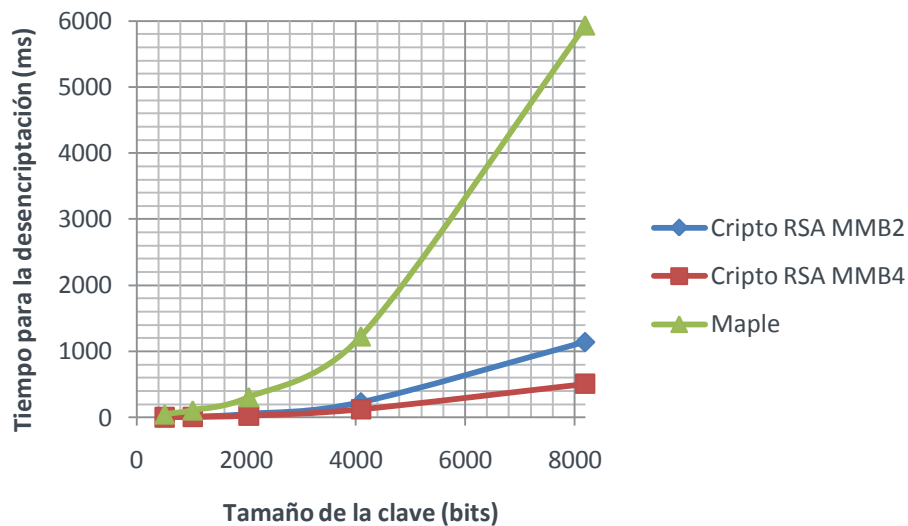


Figura 4.36. Tiempo de ejecución para la descriptación usando Maple, y los Criptoprocesadores RSA basados en el *MMB2* y el *MMB4*.

En la Figura 4.37 se presenta el área de los dos multiplicadores y los dos criptoprocesadores RSA implementados, para diferentes tamaños de clave, usando los datos de las Tablas 4.1 - 4.4. Desde esta figura se puede observar que el área del *MMB2* con respecto al área del criptoprocesador es menor, mientras que el área del *MMB4* con respecto al área del criptoprocesador es aproximadamente la mitad. Por ejemplo, para la clave de 8192 bits, el área del multiplicador es menor que el área del criptoprocesador en un factor de 1.31 y

2.27, para el *MMB2* y el *MMB4*, respectivamente. Entonces, es importante concluir que la diferencia entre estos factores se debe a que el compilador sintetiza una mayor cantidad de recursos para el criptoprocador RSA basado en el *MMB4* con el propósito no reducir la frecuencia de operación del diseño.

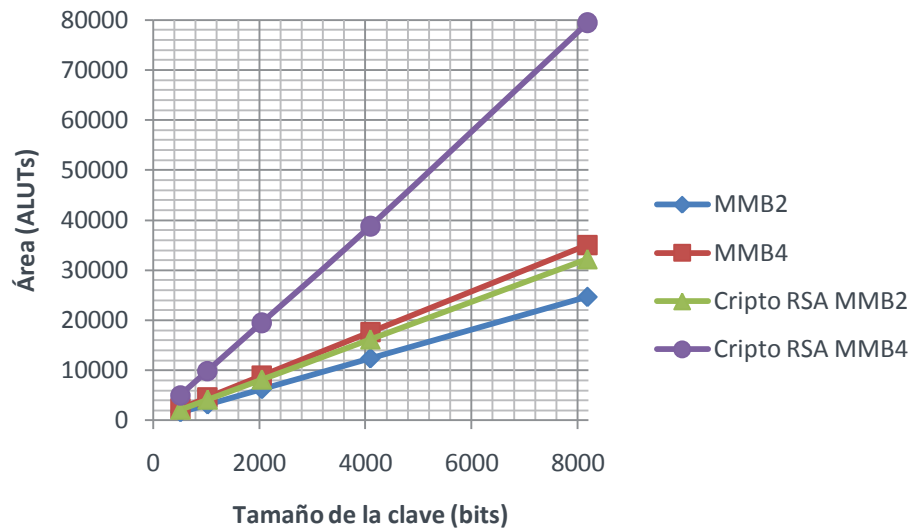


Figura 4.37. Área vs tamaño de clave para el *MMB2*, el *MMB4* y los dos criptoprocadores RSA.

En la Figura 4.38 se muestra la Frecuencia máxima de operación para el *MMB2*, el *MMB4* y los criptoprocadores basados en estos multiplicadores, para diferentes tamaños de clave, usando los datos de las Tablas 4.1 - 4.4. Desde esta figura se puede observar que la Frecuencia máxima de operación es mayor para los multiplicadores que los criptoprocadores, y la diferencia es mayor para el caso del *MMB2*. Por ejemplo, para la clave de 8192 bits, esta diferencia es de un factor de 1.57 y 1.14, para el *MMB2* y el *MMB4*, respectivamente.

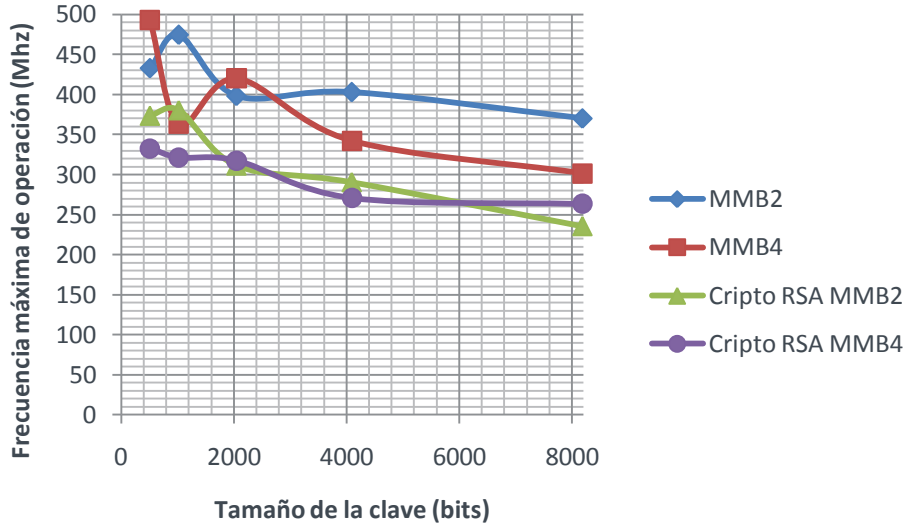


Figura 4.38. Frecuencia máxima de operación vs tamaño de clave para el *MMB2*, el *MMB4* y los dos criptoprosesadores RSA.

La Tabla 4.8 presenta los resultados de síntesis y desempeño obtenidos en este trabajo y otros presentados en la literatura para criptoprosesadores de 1024 bits. En este caso, es importante mencionar que no es posible hacer una comparación real entre nuestro trabajo y otros debido a que las implementaciones son llevadas a cabo usando diferentes arquitecturas de hardware, herramientas software, tecnologías de hardware y métricas. Sin embargo, la arquitectura sistólica permite sintetizar el criptoprosesador RSA de 8192 bits en el FPGA Stratix III, y su principal desventaja es que se reduce el “throughput”. El “throughput”, definido como Tamaño del dato de salida/Tiempo de ejecución de la operación, se calcula usando la ecuación (4.1), donde n es el tamaño de la clave y T_{EXP} es el tiempo para llevar a cabo la exponenciación modular.

$$Thr_{Crypt} = \frac{n}{T_{EXP}} \quad (4.1)$$

TABLA 4.8: RESULTADOS DE SÍNTESIS Y “THROUGHPUT” PARA
CRIPTOPROCESADORES RSA DE 1024 BITS

Referencia	r	Tecnol.	Área	Thr (Kbps)
[2]	2	Virtex 5	2982 Slices	343.2
	4		4060 Slices	503.6
[3]	2	XC2V6000	15286 Slices	11850
[4]	2	Virtex 2	5321 LUT	150
Este Trabajo	2	Stratix III	3441 ALUTs	86.71
	4		9827 ALUTs	156.1

CAPÍTULO 5

CONCLUSIONES Y TRABAJO FUTURO

En este capítulo se presentan las conclusiones de esta tesis y el trabajo futuro.

5.1. CONCLUSIONES

Esta tesis presenta el diseño de Criptoprocesadores RSA basados en multiplicadores de Montgomery base 2 y 4, para tamaños de clave de 512, 1024, 2048, 4096 y 8192 bits.

En los criptoprocesadores RSA, el bloque funcional más importante es el multiplicador modular. En este caso, se diseñaron multiplicadores de Montgomery base 2 y 4, usando una arquitectura sistólica, con el propósito de implementar en hardware estos multiplicadores de gran tamaño (por ejemplo: 4096, 8192 bits). El arreglo sistólico permite usar sólo señales locales que facilitan el ruteo de muchos elementos de procesamiento que procesan datos de gran tamaño, y reducir el retardo de la trayectoria crítica (*critical path*). En otras palabras, el arreglo sistólico es la arquitectura digital que permite sintetizar en hardware (*FPGA*) unidades aritméticas de alta complejidad desde el punto de vista del tamaño de los datos.

Adicionalmente, los multiplicadores de Montgomery base 2 y 4 llevan a cabo simultáneamente dos productos, lo cual es una ventaja para realizar la exponenciación modular usando el algoritmo de exponenciación binaria.

Los criptoprocesadores diseñados permiten realizar la encriptación y desencriptación usando el algoritmo de la exponenciación binaria, el cual es

implementado en la Unidad de control. Este algoritmo para la exponenciación modular realiza el cuadrado y el producto modular, los cuales son llevados a cabo simultáneamente usando los multiplicadores diseñados. Por lo tanto, el *throughput* de los procesadores es mayor con este algoritmo que usando otros para la exponenciación modular. Adicionalmente el algoritmo de exponenciación binaria requiere menos recursos de área, debido a que no demanda hardware para realizar precálculos, lo cual es conveniente para diseñar procesadores con datos de gran tamaño.

Los criptoprosesadores fueron diseñados usando VHDL estructural genérico, y sintetizados con Quartus II v.11 en el FPGA EP3SL150F1152C2. Teniendo en cuenta los resultados de síntesis, se puede concluir que el Criptoprosesador RSA basado en el multiplicador de Montgomery base 2 requiere menos recursos que el Criptoprosesador RSA basado en el multiplicador de Montgomery base 4, para los diferentes tamaños de clave; sin embargo este último criptoprosesador lleva a cabo en menor tiempo la exponenciación modular. Por lo tanto, el primer criptoprosesador es adecuado para implementar un criptosistema RSA embebido en un FPGA, mientras el segundo permite obtener un mayor *throughput* usando más recursos.

Adicionalmente, este trabajo presenta una comparación entre las implementaciones hardware y software para llevar a cabo la exponenciación modular. Desde esta comparación es posible concluir que los criptoprosesadores diseñados requieren menor tiempo para realizar la exponenciación modular que su contraparte en *software* usando Maple.

Teniendo en cuenta los resultados de síntesis y de verificación en *hardware* se puede concluir que los criptoprosesadores diseñados pueden ser usados en el diseño de un Criptosistema RSA embebido en un SoC.

5.2. TRABAJO FUTURO

El trabajo futuro estará orientado a mejorar el *throughput* y disminuir la cantidad de recursos de hardware de los criptoprocesadores RSA realizando las siguientes actividades:

- Optimizar la arquitectura del criptoprocesador RSA de 8192 bits basado en el multiplicador de Montgomery base 2, reduciendo el número de registros usados y bloques combinacionales.
- Diseñar el multiplicador de Montgomery base 4 de 8192 bits usando codificación de Booth, lo cual evita los dos precálculos $3*B$ y reduce la cantidad de recursos del Criptoprocesador.
- Diseñar los multiplicadores de Montgomery base 2 y 4 de 8192 bits usando diferentes tamaños de palabra, por ejemplo: 8, 16, 32 bits, con el fin de mejorar el *throughput* de los criptoprocesadores RSA.
- Diseñar el criptoprocesador RSA de 8192 bits basado en el multiplicador de Montgomery base 8, con el propósito de obtener un mejor *throughput*.
- Diseñar los multiplicadores de Montgomery base 2, 4 y 8 de 8192 bits, usando un arreglo semi-sistólico con el propósito de mejorar el *throughput* de los criptoprocesadores.

BIBLIOGRAFÍA

- [1] F. Rodríguez, N. Saqib, A. Diaz-Perez y C. Kaya. "Cryptographic Algorithms on Reconfigurable Hardware", Springer Series on Signals and Communications Technology, 2006, pp. 92-138.
- [2] G. Sutter, J. Deschamps and J. Imaña. "Modular Multiplication and Exponentiation Architectures for Fast RSA Cryptosystem Based on Digit Serial Computation", IEEE Transactions on Industrial Electronics, Vol. 58, No. 7, July 2011.
- [3] M. Shieh, J. Chen, W. Lin and H. Wu, "*A new Algorithm for High-Speed Modular Multiplication Design*", IEEE Transactions on Circuits and Systems, Vol 56, No. 9, September 2009.
- [4] J. Liu and J. Dong, "*Design and Implementation of an Efficient RSA Cryptoprocessor*", IEEE International conference on Progress Informatics and Computing, December 2010.
- [5] Erkey Savaş, Çetin Kaya Koç, "Finite Field Arithmetic for Cryptography", IEEE Circuits and Systems Magazine, pp 40 -56.
- [6] <http://spectrum.ieee.org/telecom/security/too-big-to-hack> IEEE Spectrum. Torturing the Secret out of a Secure Chip.
- [7] <http://www.rsa.com/rsalabs/node.asp?id=2146>
- [8] <http://www.rsa.com/rsalabs/node.asp?id=2125>. PKCS #1 V2.1.

[9] F. Bernard. "Scalable hardware implenting high-radix Montgomery multiplication algorithm", ScienceDirect, Journal of Systems Architecture 53 (2007), pp 117-126.

[10] K. J. Lee, K. Y. Yoo, "Systolic multiplier for Montgomery's algorithm", Integration, the VLSI journal, 2002, 32(1-2), pp. 31-35.

[11] A Memory Optimized Public-Key Crypto Algorithm Using Modify Modular Exponentiation (MME).

[12] P. Amberg, N. Pinckney, D. Money. "Parallel High-Radix Montgomery Multipliers", Conference on Signals, Systems and Computers, 2008, pp. 772-776.

[13] F. Gang. "Design of Modular Multiplier Based on Improved Montgomery Algorithm and Systolic Array", First International Multi-Symposiums on Computer and Computational Sciences, IEEE proceedings, 2006.

[14] http://www.altera.com/literature/hb/qts/qts_qii53009.pdf "Design Debugging Using the SignalTap II Logic Analyzer", Quartus II Handbook Version 11.1, Volume 3, November 2011.

[15] <http://www.rsa.com/rsalabs/node.asp?id=2125>, "PKCS # 1: RSA Cryptography Standard", Public Key Cryptography Standards, 2009.

ANEXO 1

DATOS PARA REALIZAR LA PRUEBA DE

LA EXPONENCIACIÓN MODULAR DE 8192 BITS

M

208877776282630501338350542143324876515992849809476756076846696656
790781594311491369765362386999511668178021342887852567597514687637
158721452647217570273055589191395308741999668072318026876743662885
614002371189245275263767879542549134466936868917323499361581741160
740814256809748023721822893595556719605801337387795376357557189381
126038052188119915890686564693860605339288611805003194479973542843
108338767111977058297263647582886899346817562374527899295020037808
269801683412335018733666770110206594417653910153996723273882386603
042759365167437618366731350244263698569273625110045199660082468956
972519134898438923404761301182649122165146367076017521724420566854
039539640462633803535601335039097015984327283874057075024956802981
431827091996558102679922310358854221366226818116854576855958310969
956590864706903413044653812278981197538600424592679137556575789688
123201482589134983964610114328475430963264276126209180583227385341
668571288146089579994380356293152694644770053450611979959199218159
893840354924963543689973491131850035665814094623886633110161513644
369314349274659376982563904063491400488185323382174829677015682385
960904596371467795529620625217180600260482693437945334643298302226
3205841563476066872180487609416680806308380671

expn

748866360096423681622274881354738897450156535565260420552199680967
572444246069902603836708447092532601340000257043027380363381624266
140309101175592587072100445818447164255871547352053501810864182098
462137268431465712828195166124067490489763692716333921544927132266
772982781882195282267915580459647088427133168847197211339964842646

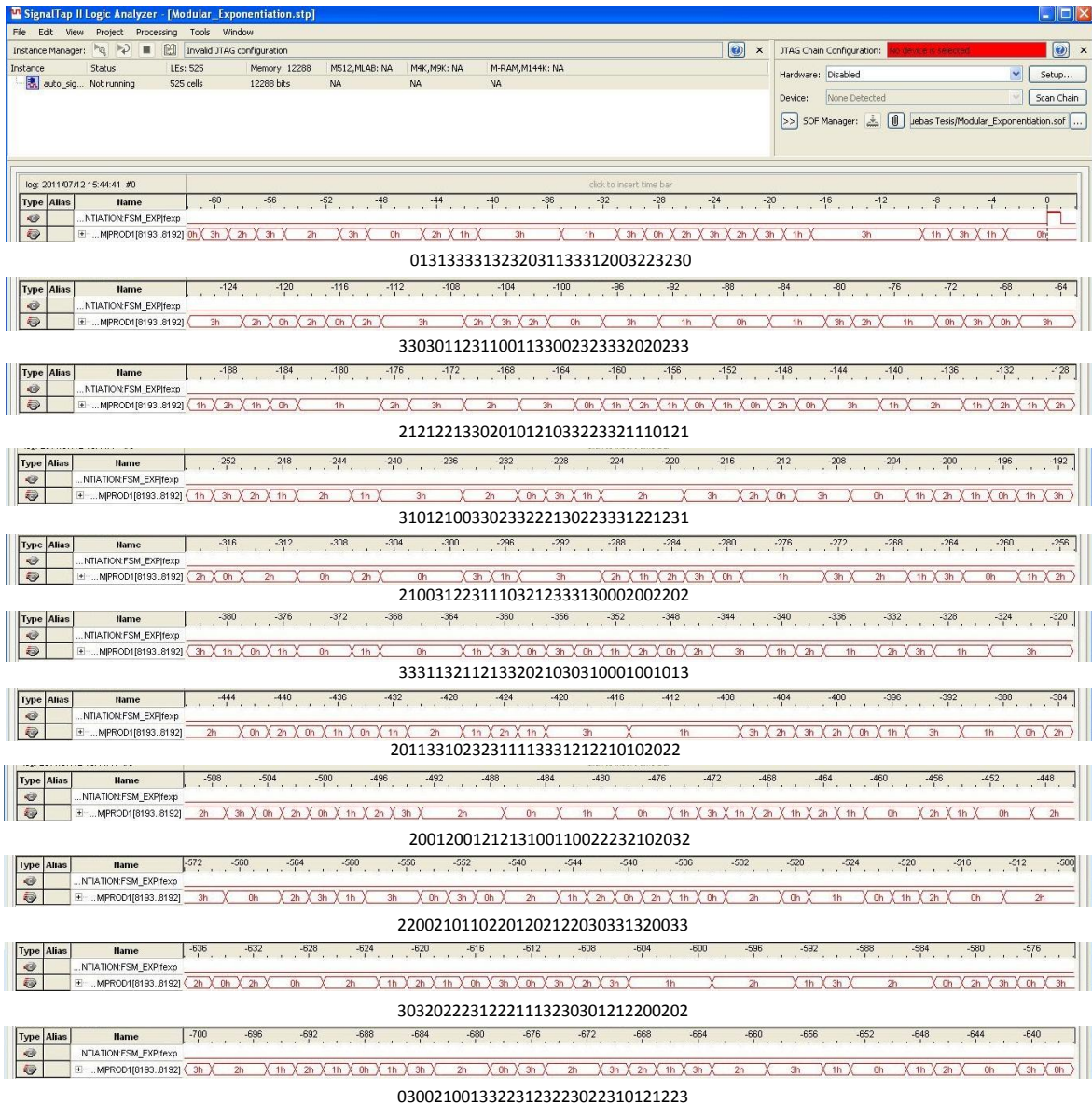
549425471466597244007327159480356777011074268100125518777502682288
 879327013543630828615131793606586674893330852782711664424415950416
 636860844114055911911120833198424017256629136194125823106745516439
 064396859727696540459616135767452142492799925753680998932537671013
 705171143212745934223080710584047520469019555699148478834405589620
 399925113953172213508872143130201482641176427521838552336729338336
 075116315382263840469889125848421803419704957765713597165737939764
 064059184689907943264345757408663073353053446142576140065114604969
 073657375223331115144872456469444835662305831051359362460638245053
 470558241917703974505412228223037208785135157531400156995130390642
 596483000313504011703726161324100783001276352559454675518667267925
 299450810674338150187446441874953124922404096799445318854435437309
 316821487004870193401471632184155503609960964157177592201163021589
 385640680356553781050388973169233129035184760424683612894607571484
 444326594649414988811022881881368144961882540489632025176437041975
 524751039864272317557717161631891916981943381729585020834566794118
 665518166949753972727918515449127136767488781842965424020987968007
 280785987449646980863917215923178786900920327368363164193763550097
 309740245994206333513268466778255471789529245128376729185094363907
 354063896221073446431341791618147554194982289316986578393981433857
 858985316959235687133096789561830712009408861835531025501578297387
 067100106747342957908646792196450695676024903073983591324419147842
 390084318483883102734461235234376606760046223322649834836050052674
 845599800050546430391569462612511511410316667690727078337846031467
 936394726359471591200901167599913698302957381925065425684471456593
 414355639253045422208336941690945439351937119262239046267895537095
 454667437764468622174579189906375274201992949548589298797176945413
 549985317917056056542955259350429479847755312154454149811655504376
 947816170458641493124380468180004152284911157845350518885149174510
 041898967899177524972288958705297574379044120282667178809960794400
 024446030056005641295505513863618149757725506761020314726428324161
 052734960092470646607525771284937646773788887871233592693009450385
 258002012916012937292852

PQ

$$2^{8191} + 1$$

ANEXO 2

VERIFICACIÓN HARDWARE USANDO SIGNALTAP DEL CRIPTOPROCEESADOR RSA DE 8192 BITS



Type	Alias	Name	-764	-760	-756	-752	-748	-744	-740	-736	-732	-728	-724	-720	-716	-712	-708	-704
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	1h	3h	1h	2h	0h	1h	0h	1h	0h	3h	0h	1h	3h	0h	3h	1h

23120233332120130031030101021131

Type	Alias	Name	-828	-824	-820	-816	-812	-808	-804	-800	-796	-792	-788	-784	-780	-776	-772	-768
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	3h	1h	2h	0h	2h	3h	0h	1h	0h	1h	0h	2h	1h	0h	2h	1h

1120010211123222220110103202133

Type	Alias	Name	-892	-888	-884	-880	-876	-872	-868	-864	-860	-856	-852	-848	-844	-840	-836	-832
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	0h	1h	2h	1h	3h	2h	1h	0h	1h	3h	1h	2h	0h	1h	2h	1h

12302220102100211311000112312210

Type	Alias	Name	-956	-952	-948	-944	-940	-936	-932	-928	-924	-920	-916	-912	-908	-904	-900	-896
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	3h	2h	1h	3h	0h	1h	3h	2h	1h	2h	3h	2h	3h	0h	2h	0h

2223320220203332332212310312333

Type	Alias	Name	-1020	-1016	-1012	-1008	-1004	-1000	-996	-992	-988	-984	-980	-976	-972	-968	-964	-960
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	0h	3h	0h	1h	3h	1h	3h	2h	0h	3h	1h	2h	1h	2h	0h	1h

11013311103030021211300023131030

Type	Alias	Name	-1084	-1080	-1076	-1072	-1068	-1064	-1060	-1056	-1052	-1048	-1044	-1040	-1036	-1032	-1028	-1024
		...M_EXPfexp																
		⊕ ...3..8192]	3h	0h	3h	2h	1h	2h	1h	2h	3h	1h	3h	1h	0h	3h	0h	1h

01200110021030001313221121223003

Type	Alias	Name	-1148	-1144	-1140	-1136	-1132	-1128	-1124	-1120	-1116	-1112	-1108	-1104	-1100	-1096	-1092	-1088
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	0h	2h	0h	3h	2h	3h	0h	3h	1h	0h	2h	0h	3h	1h	2h	1h

33231013310121300201300332300220

Type	Alias	Name	-1212	-1208	-1204	-1200	-1196	-1192	-1188	-1184	-1180	-1176	-1172	-1168	-1164	-1160	-1156	-1152
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	1h	2h	0h	1h	3h	2h	1h	2h	3h	2h	3h	1h	3h	1h	0h	2h

32233130012000013132132321231021

Type	Alias	Name	-1276	-1272	-1268	-1264	-1260	-1256	-1252	-1248	-1244	-1240	-1236	-1232	-1228	-1224	-1220	-1216
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	3h	2h	0h	2h	1h	2h	1h	2h	3h	1h	2h	3h	2h	3h	0h	1h

0331132302310323021133212120233

Type	Alias	Name	-1340	-1336	-1332	-1328	-1324	-1320	-1316	-1312	-1308	-1304	-1300	-1296	-1292	-1288	-1284	-1280
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	1h	0h	1h	3h	2h	0h	3h	2h	3h	0h	2h	1h	0h	3h	2h	1h

31110222230110120323022331110011

Type	Alias	Name	-1404	-1400	-1396	-1392	-1388	-1384	-1380	-1376	-1372	-1368	-1364	-1360	-1356	-1352	-1348	-1344
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	1h	3h	0h	1h	2h	1h	3h	1h	3h	2h	0h	3h	2h	3h	2h	0h

02323310121232330232313122100331

Type	Alias	Name	-1468	-1464	-1460	-1456	-1452	-1448	-1444	-1440	-1436	-1432	-1428	-1424	-1420	-1416	-1412	-1408
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	3h	0h	2h	1h	3h	1h	2h	0h	1h	0h	1h	2h	3h	0h	3h	1h

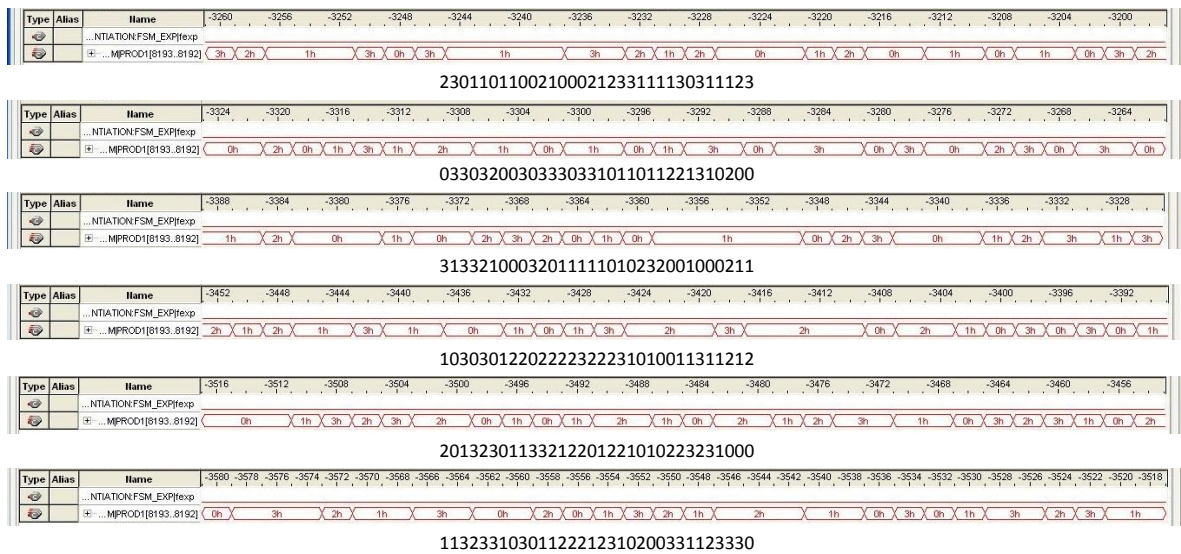
21311332132303210111002213311203

Type	Alias	Name	-1532	-1528	-1524	-1520	-1516	-1512	-1508	-1504	-1500	-1496	-1492	-1488	-1484	-1480	-1476	-1472
		...NTIATIONFSM_EXPfexp																
		⊕ ...MPROD1[8193..8192]	1h	3h	1h	2h	1h	3h	1h	2h	1h	0h	1h	0h	3h	2h	1h	0h

10033100332001230100012131221311

Type	Alias	Name	-1596	-1592	-1588	-1584	-1580	-1576	-1572	-1568	-1564	-1560	-1556	-1552	-1548	-1544	-1540	-1536
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	3h	2h	3h	1h	0h	3h	2h	0h	1h	2h	3h	2h	1h	2h	0h
23033132002102123210230001322331																		
Type	Alias	Name	-1660	-1656	-1652	-1648	-1644	-1640	-1636	-1632	-1628	-1624	-1620	-1616	-1612	-1608	-1604	-1600
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	2h	1h	2h	0h	3h	1h	2h	3h	2h	1h	3h	0h	1h	2h	1h
221023313121110312322213022122211																		
Type	Alias	Name	-1724	-1720	-1716	-1712	-1708	-1704	-1700	-1696	-1692	-1688	-1684	-1680	-1676	-1672	-1668	-1664
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	3h	1h	2h	0h	3h	2h	1h	0h	2h	1h	3h	1h	0h	1h	3h	1h
10101011310013312012200012300213																		
Type	Alias	Name	-1788	-1784	-1780	-1776	-1772	-1768	-1764	-1760	-1756	-1752	-1748	-1744	-1740	-1736	-1732	-1728
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	3h	0h	3h	1h	2h	3h	0h	2h	0h	1h	0h	3h	2h	0h	2h	1h
10331033202002023001020303221303																		
Type	Alias	Name	-1852	-1848	-1844	-1840	-1836	-1832	-1828	-1824	-1820	-1816	-1812	-1808	-1804	-1800	-1796	-1792
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	0h	2h	3h	1h	2h	0h	2h	0h	2h	0h	3h	0h	2h	0h	1h	0h
13021302330102032223020202113220																		
Type	Alias	Name	-1916	-1912	-1908	-1904	-1900	-1896	-1892	-1888	-1884	-1880	-1876	-1872	-1868	-1864	-1860	-1856
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	0h	1h	3h	2h	3h	2h	1h	0h	3h	2h	3h	0h	2h	3h	1h	0h
33333120013033132030323012332310																		
Type	Alias	Name	-1980	-1976	-1972	-1968	-1964	-1960	-1956	-1952	-1948	-1944	-1940	-1936	-1932	-1928	-1924	-1920
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	2h	0h	1h	3h	2h	0h	2h	0h	1h	3h	1h	0h	3h	2h	0h	3h
33222031230213223013102023111022																		
Type	Alias	Name	-2044	-2040	-2036	-2032	-2028	-2024	-2020	-2016	-2012	-2008	-2004	-2000	-1996	-1992	-1988	-1984
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	3h	1h	2h	3h	0h	1h	2h	0h	1h	3h	2h	0h	3h	0h	2h
21022120300331033213102103211131																		
Type	Alias	Name	-2108	-2104	-2100	-2096	-2092	-2088	-2084	-2080	-2076	-2072	-2068	-2064	-2060	-2056	-2052	-2048
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	3h	0h	2h	1h	2h	0h	2h	3h	0h	2h	3h	0h	2h	0h	3h
0331303011002032203220221203111																		
Type	Alias	Name	-2172	-2168	-2164	-2160	-2156	-2152	-2148	-2144	-2140	-2136	-2132	-2128	-2124	-2120	-2116	-2112
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	3h	0h	1h	0h	2h	1h	0h	3h	0h	2h	3h	1h	0h	2h	0h	2h
11220120020200013313203031200103																		
Type	Alias	Name	-2236	-2232	-2228	-2224	-2220	-2216	-2212	-2208	-2204	-2200	-2196	-2192	-2188	-2184	-2180	-2176
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	3h	2h	1h	0h	1h	3h	2h	3h	1h	0h	1h	0h	2h	1h	3h
10312131112013220010132231012311																		
Type	Alias	Name	-2300	-2296	-2292	-2288	-2284	-2280	-2276	-2272	-2268	-2264	-2260	-2256	-2252	-2248	-2244	-2240
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	1h	0h	3h	1h	3h	0h	3h	1h	3h	0h	2h	3h	2h	0h	3h	0h
21030330302320202113331130031301																		
Type	Alias	Name	-2364	-2360	-2356	-2352	-2348	-2344	-2340	-2336	-2332	-2328	-2324	-2320	-2316	-2312	-2308	-2304
		...NTIATIONFSM_EXPFexp																
		± ...MPROD1[8193..8192]	0h	3h	0h	2h	1h	3h	1h	2h	1h	3h	2h	3h	0h	1h	2h	0h
23102302223312100323312131200300																		

Type	Alias	Name	-2428	-2424	-2420	-2416	-2412	-2408	-2404	-2400	-2396	-2392	-2388	-2384	-2380	-2376	-2372	-2368
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	3h	2h	0h	1h	0h	2h	3h	0h	1h	2h	3h	1h	0h	3h	0h	2h
00333320033301321331111003201023																		
Type	Alias	Name	-2492	-2488	-2484	-2480	-2476	-2472	-2468	-2464	-2460	-2456	-2452	-2448	-2444	-2440	-2436	-2432
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	2h	1h	2h	0h	1h	0h	1h	2h	3h	0h	1h	2h	3h	1h	0h	2h
11203331120300013103221101102112																		
Type	Alias	Name	-2556	-2552	-2548	-2544	-2540	-2536	-2532	-2528	-2524	-2520	-2516	-2512	-2508	-2504	-2500	-2496
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	1h	3h	0h	3h	1h	2h	0h	3h	1h	3h	0h	2h	1h	3h	0h	1h
31201021132210331203133021333031																		
Type	Alias	Name	-2620	-2616	-2612	-2608	-2604	-2600	-2596	-2592	-2588	-2584	-2580	-2576	-2572	-2568	-2564	-2560
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	2h	0h	1h	0h	1h	3h	1h	3h	1h	0h	1h	3h	0h	2h	3h	0h
02223023113031303203100131310102																		
Type	Alias	Name	-2684	-2680	-2676	-2672	-2668	-2664	-2660	-2656	-2652	-2648	-2644	-2640	-2636	-2632	-2628	-2624
		...EXPfexp																
		MPROD1[8192]	2h	3h	0h	1h	2h	0h	1h	0h	1h	2h	0h	2h	1h	0h	2h	0h
11002020302012021111011000221032																		
Type	Alias	Name	-2748	-2744	-2740	-2736	-2732	-2728	-2724	-2720	-2716	-2712	-2708	-2704	-2700	-2696	-2692	-2688
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	2h	0h	1h	3h	2h	1h	2h	1h	0h	1h	3h	0h	2h	0h	3h	0h
01213213000003020310001221231022																		
Type	Alias	Name	-2812	-2808	-2804	-2800	-2796	-2792	-2788	-2784	-2780	-2776	-2772	-2768	-2764	-2760	-2756	-2752
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	3h	1h	2h	1h	0h	1h	3h	2h	1h	0h	2h	1h	3h	2h	0h	1h
30012102310211231312012231011213																		
Type	Alias	Name	-2876	-2872	-2868	-2864	-2860	-2856	-2852	-2848	-2844	-2840	-2836	-2832	-2828	-2824	-2820	-2816
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	1h	0h	2h	1h	0h	2h	1h	3h	2h	1h	3h	0h	3h	1h	3h	0h
10321300113130312130233321111201																		
Type	Alias	Name	-2940	-2936	-2932	-2928	-2924	-2920	-2916	-2912	-2908	-2904	-2900	-2896	-2892	-2888	-2884	-2880
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	1h	0h	1h	0h	3h	0h	3h	1h	2h	0h	2h	0h	3h	0h	2h	1h
10232320211222001030202130300101																		
Type	Alias	Name	-3004	-3000	-2996	-2992	-2988	-2984	-2980	-2976	-2972	-2968	-2964	-2960	-2956	-2952	-2948	-2944
		...M_EXPTexp																
		MPROD1[8192]	2h	0h	1h	0h	3h	0h	2h	1h	3h	0h	1h	0h	1h	3h	0h	1h
31311031001003311200030010002222																		
Type	Alias	Name	-3068	-3064	-3060	-3056	-3052	-3048	-3044	-3040	-3036	-3032	-3028	-3024	-3020	-3016	-3012	-3008
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	1h	0h	1h	0h	1h	0h	1h	2h	0h	3h	2h	3h	0h	3h	0h	3h
33130300323333021100111110001001																		
Type	Alias	Name	-3132	-3128	-3124	-3120	-3116	-3112	-3108	-3104	-3100	-3096	-3092	-3088	-3084	-3080	-3076	-3072
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	1h	3h	0h	3h	1h	0h	2h	1h	2h	1h	2h	3h	0h	1h	0h	2h
03120323001003211121021120113031																		
Type	Alias	Name	-3196	-3192	-3188	-3184	-3180	-3176	-3172	-3168	-3164	-3160	-3156	-3152	-3148	-3144	-3140	-3136
		...NTIATIONFSM_EXPTexp																
		MPROD1[8193..8192]	2h	0h	1h	2h	3h	2h	0h	3h	0h	2h	1h	3h	0h	1h	2h	3h
22333211333110011031120330232102																		



Resultado verificado en el FPGA (hexadecimal):

01(1)31(D)33(F)33(F)13(7)23(B)20(8)31(D)13(7)33(F)12(6)00(0)32(E)23(B)23(B)0
3(3)30(C)30(C)11(5)23(B)11(5)00(0)11(5)33(F)00(0)23(B)23(B)33(F)20(8)20(8)23
(B)32(E)12(6)12(6)21(9)33(F)02(2)01(1)01(1)21(9)03(3)32(E)23(B)32(E)11(5)10(4
)12(6)13(7)10(4)12(6)10(4)03(3)30(C)23(B)32(E)22(A)13(7)02(2)23(B)33(F)12(6)2
1(9)23(B)12(6)10(4)03(3)12(6)23(B)11(5)10(4)32(E)12(6)33(F)31(D)30(C)00(0)20(
8)02(2)20(8)23(B)33(F)11(5)32(E)11(5)21(9)33(F)20(8)21(9)03(3)03(3)10(4)00(0)
10(4)01(1)01(1)32(E)01(1)13(7)31(D)02(2)32(E)31(D)11(5)13(7)33(F)12(6)12(6)21
(9)01(1)02(2)02(2)22(A)00(0)12(6)00(0)12(6)12(6)13(7)10(4)01(1)10(4)02(2)22(A)
32(E)10(4)20(8)32(E)22(A)00(0)21(9)01(1)10(4)22(A)01(1)20(8)21(9)22(A)03(3)0
3(3)31(D)32(E)00(0)33(F)30(C)32(E)02(2)22(A)31(D)22(A)21(9)11(5)32(E)30(C)3
0(C)12(6)12(6)20(8)02(2)02(2)03(3)00(0)21(9)00(0)13(7)32(E)23(B)12(6)32(E)23(
B)02(2)23(B)10(4)12(6)12(6)23(B)23(B)12(6)02(2)33(F)33(F)21(9)20(8)13(7)00(0)
31(D)03(3)01(1)01(1)02(2)11(5)31(D)11(5)20(8)01(1)02(2)11(5)12(6)32(E)22(A)22
(A)20(8)11(5)01(1)03(3)20(8)21(9)33(F)12(6)30(C)22(A)20(8)10(4)21(9)00(0)21(9)

13(7)11(5)00(0)01(1)12(6)31(D)22(A)10(4)22(A)23(B)32(E)02(2)20(8)20(8)33(F)3
3(F)23(B)32(E)21(9)23(B)10(4)31(D)23(B)33(F)11(5)01(1)33(F)11(5)10(4)30(C)30
(C)02(2)12(6)11(5)30(C)00(0)23(B)13(7)10(4)30(C)01(1)20(8)01(1)10(4)02(2)10(4
)30(C)00(0)13(7)13(7)22(A)11(5)21(9)22(A)30(C)03(3)33(F)23(B)10(4)13(7)31(D)
01(1)21(9)30(C)02(2)01(1)30(C)03(3)32(E)30(C)02(2)20(8)32(E)23(B)31(D)30(C)0
1(1)20(8)00(0)01(1)31(D)32(E)13(7)23(B)21(9)23(B)10(4)21(9)03(3)31(D)13(7)23(
B)02(2)31(D)03(3)23(B)20(8)21(9)13(7)32(E)12(6)12(6)02(2)33(F)31(D)11(5)02(2)
22(A)23(B)01(1)10(4)12(6)03(3)23(B)02(2)23(B)31(D)11(5)00(0)11(5)02(2)32(E)3
3(F)10(4)12(6)12(6)32(E)33(F)02(2)32(E)31(D)31(D)22(A)10(4)03(3)31(D)21(9)31
(D)13(7)32(E)13(7)23(B)03(3)21(9)01(1)11(5)00(0)22(A)13(7)31(D)12(6)03(3)10(4
)03(3)31(D)00(0)33(F)20(8)01(1)23(B)01(1)00(0)01(1)21(9)31(D)22(A)13(7)11(5)2
3(B)03(3)31(D)32(E)00(0)21(9)02(2)12(6)32(E)10(4)23(B)00(0)01(1)32(E)23(B)31
(D)22(A)10(4)23(B)31(D)31(D)21(9)10(4)31(D)23(B)22(A)21(9)30(C)22(A)12(6)22
(A)11(5)10(4)10(4)10(4)11(5)31(D)00(0)13(7)31(D)20(8)12(6)20(8)00(0)12(6)30(C
)02(2)13(7)10(4)33(F)10(4)33(F)20(8)20(8)02(2)02(2)30(C)01(1)02(2)03(3)03(3)22
(A)13(7)03(3)13(7)02(2)13(7)02(2)33(F)01(1)02(2)03(3)22(A)23(B)02(2)02(2)02(2)
11(5)32(E)20(8)33(F)33(F)31(D)20(8)01(1)30(C)33(F)13(7)20(8)30(C)32(E)30(C)1
2(6)33(F)23(B)10(4)33(F)22(A)20(8)31(D)23(B)02(2)13(7)22(A)30(C)13(7)10(4)20
(8)23(B)11(5)10(4)22(A)21(9)02(2)21(9)20(8)30(C)03(3)31(D)03(3)32(E)13(7)10(4
)21(9)03(3)21(9)11(5)31(D)03(3)31(D)30(C)30(C)11(5)00(0)20(8)32(E)20(8)32(E)2
0(8)22(A)21(9)20(8)31(D)11(5)11(5)22(A)01(1)20(8)02(2)02(2)00(0)01(1)33(F)13(
7)20(8)30(C)31(D)20(8)01(1)03(3)10(4)31(D)21(9)31(D)11(5)20(8)13(7)22(A)00(0)
10(4)13(7)22(A)31(D)01(1)23(B)11(5)21(9)03(3)03(3)30(C)30(C)23(B)20(8)20(8)2
1(9)13(7)33(F)11(5)30(C)03(3)13(7)01(1)23(B)10(4)23(B)02(2)22(A)33(F)12(6)10(
4)03(3)23(B)31(D)21(9)31(D)20(8)03(3)00(0)00(0)33(F)33(F)20(8)03(3)33(F)01(1)
32(E)13(7)31(D)11(5)10(4)03(3)20(8)10(4)23(B)11(5)20(8)33(F)31(D)12(6)03(3)00
(0)01(1)31(D)03(3)22(A)11(5)01(1)10(4)21(9)12(6)31(D)20(8)10(4)21(9)13(7)22(A
)10(4)33(F)12(6)03(3)13(7)30(C)21(9)33(F)30(C)31(D)02(2)22(A)30(C)23(B)11(5)
30(C)31(D)30(C)32(E)03(3)10(4)01(1)31(D)31(D)01(1)02(2)11(5)00(0)20(8)20(8)3
0(C)20(8)12(6)02(2)11(5)11(5)01(1)10(4)00(0)22(A)10(4)32(E)01(1)21(9)32(E)13(
7)00(0)00(0)03(3)02(2)03(3)10(4)00(0)12(6)21(9)23(B)10(4)22(A)30(C)01(1)21(9)
02(2)31(D)02(2)11(5)23(B)13(7)12(6)01(1)22(A)31(D)01(1)12(6)13(7)10(4)32(E)1
3(7)00(0)11(5)31(D)30(C)31(D)21(9)30(C)23(B)33(F)21(9)11(5)12(6)01(1)10(4)23(
B)23(B)20(8)21(9)12(6)22(A)00(0)10(4)30(C)20(8)21(9)30(C)30(C)01(1)01(1)31(D
)31(D)10(4)31(D)00(0)10(4)03(3)31(D)12(6)00(0)03(3)00(0)10(4)00(0)22(A)22(A)3
3(F)13(7)03(3)00(0)32(E)33(F)33(F)02(2)11(5)00(0)11(5)11(5)10(4)00(0)10(4)01(1
)03(3)12(6)03(3)23(B)00(0)10(4)03(3)21(9)11(5)21(9)02(2)11(5)20(8)11(5)30(C)31
(D)22(A)33(F)32(E)11(5)33(F)31(D)10(4)01(1)10(4)31(D)12(6)03(3)30(C)23(B)21(
9)02(2)23(B)01(1)10(4)11(5)00(0)21(9)00(0)02(2)12(6)33(F)11(5)11(5)30(C)31(D)
11(5)23(B)03(3)30(C)32(E)00(0)30(C)33(F)30(C)33(F)10(4)11(5)01(1)12(6)21(9)3
1(D)02(2)00(0)31(D)33(F)21(9)00(0)03(3)20(8)11(5)11(5)10(4)10(4)23(B)20(8)01(

1)00(0)02(2)11(5)10(4)30(C)30(C)12(6)20(8)22(A)22(A)32(E)22(A)31(D)01(1)00(0)
)11(5)31(D)12(6)12(6)20(8)13(7)23(B)01(1)13(7)32(E)12(6)20(8)12(6)21(9)01(1)0
2(2)23(B)23(B)10(4)00(0)11(5)32(E)33(F)10(4)30(C)11(5)22(A)21(9)23(B)10(4)20
(8)03(3)31(D)12(6)33(F)30(C)